1503.68591

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re U.S. Patent Application)
)
Applicant: Makoto Nakanishi)
)
Serial No.)
)
Filed: October 24, 2003)
)
For: PARALLEL PROCESSING)
METHOD FOR INVERSE)
MATRIX FOR SHARED)
MEMORY TYPE SCALAR)
PARALLEL COMPUTER)

I hereby certify that this paper is being deposited with the United States Postal Service as EXPRESS MAIL in an envelope addressed to: Mail Stop PATENT APPLICATION, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on this date.

Oct. 24, 2003 *Date*

Express Mail Label No.: EV032735309US

CLAIM FOR PRIORITY

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Applicant claims foreign priority benefits under 35 U.S.C. § 119 on the basis of Japanese Patent Application No. 2003-074548, filed March 18, 2003. A certified copy of the priority document is enclosed.

In addition, a certified copy of Japanese Patent Application No. 2002-079909, filed March 22, 2002 in Japan, was filed in Application Serial No. 10/288,984 on November 6, 2002.

Respectfully submitted,
GREER, BURNS & CRAIN, LTD.

By
Patrick G. Burns
Registration No. 29,367

October 24, 2003
300 South Wacker Drive
Suite 2500
Chicago, Illinois 60606
Telephone: 312.360.008
Facsimile: 312.360.9315

JAPAN PATENT OFFICE

This is to certify that the annexed is a true copy of the following application as filed with this office.

Date of Application: March 18, 2003

Application Number: Patent Application No. 2003-074548
[ST.10/C] [JP2003-074548]

Applicant(s): FUJITSU LIMITED

August 18, 2003

Commissioner,
Japan Patent Office Yasuo IMAI

Certificate No. P2003-3067303

1503.68591
312.360.0010

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: ２００３年 ３月１８日

出 願 番 号
Application Number: 特願２００３－０７４５４８

[ST. 10/C]: [ＪＰ２００３－０７４５４８]

出 願 人
Applicant(s): 富士通株式会社

２００３年 ８月１８日

特許庁長官
Commissioner,
Japan Patent Office 今井康夫

出証番号 出証特２００３－３０６７３０３

【書類名】　特許願
【整理番号】　0253797
【提出日】　平成15年 3月18日
【あて先】　特許庁長官殿
【国際特許分類】　G06F 17/16
【発明の名称】　共有メモリ型スカラ並列計算機用逆行列の並列処理方法
【請求項の数】　5
【発明者】
【住所又は居所】　神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
【氏名】　中西　誠
【特許出願人】
【識別番号】　000005223
【氏名又は名称】　富士通株式会社
【代理人】
【識別番号】　100074099
【住所又は居所】　東京都千代田区二番町８番地２０　二番町ビル３Ｆ
【弁理士】
【氏名又は名称】　大菅　義之
【電話番号】　03-3238-0031
【選任した代理人】
【識別番号】　100067987
【住所又は居所】　神奈川県横浜市鶴見区北寺尾７－２５－２８－５０３
【弁理士】
【氏名又は名称】　久木元　彰
【電話番号】　045-573-3683
【先の出願に基づく優先権主張】
【出願番号】　特願2002- 79909
【出願日】　平成14年 3月22日

【手数料の表示】

【予納台帳番号】　012542

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9705047

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　共有メモリ型スカラ並列計算機用逆行列の並列処理方法

【特許請求の範囲】

【請求項１】共有メモリ型スカラ並列計算機用逆行列の並列処理方法を実現させるプログラムであって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、

該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にＬＵ分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、１つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、

を備えることを特徴とする方法を共有メモリ型スカラ並列計算機に実現させるプログラム。

【請求項２】前記共有メモリ型スカラ並列計算機は、複数のプロセッサと、該プロセッサに対応して設けられる複数のキャッシュメモリと、複数の共有メモリと、これらを通信可能なように接続する相互結合網とからなることを特徴とする請求項１に記載のプログラム。

【請求項３】上記方法は、Ｇａｕｓｓ－Ｊｏｒｄａｎの方法をブロック毎に並列に計算する構成としたことを特徴とする請求項１に記載のプログラム。

【請求項４】前記各ブロックを分割して並列計算するさいの分割の幅は、逆行列を求める行列の大きさと並列処理のために利用できるプロセッサ数から、並列処理しない正方形ブロックの演算量の総和が演算全体の１％程度となるように設定されることを特徴とする請求項１に記載のプログラム。

【請求項５】共有メモリ型スカラ並列計算機用逆行列の並列処理方法であって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、

該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にＬＵ分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、１つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、

を備えることを特徴とする方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、共有メモリ型スカラ並列計算機における演算処理に関する。

【０００２】

【従来の技術】

従来、ベクトル計算機では、ある行列の逆行列を求める場合、Ｇａｕｓｓ－Ｊｏｒｄａｎの方法をベースに、メモリアクセス系の動作が速いことを利用した計算方法を用いて行っていた。例えば、ループ内の命令を複数回分明示的に記述する２重アンローリング等の手法が用いられていた。

【０００３】

Ｇａｕｓｓ－Ｊｏｒｄａｎの方法（あるいは、単にＧａｕｓｓ法）による逆行列を求める方法を以下に説明する（なお、以下の説明ではピボットの入れ替えは省略するが実際には、ピボットの入れ替えのため行ベクトルの入れ替えを行って

いる。

まず、Ａを逆行列を求めるべき行列、ｘ、ｙは、適当な列ベクトルとする。

Ａｘ＝ｙは、行列要素をあらわに記載すると以下のような連立一次方程式となる。

$a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n=y_1$

$a_{21}x_1+a_{22}x_2+\cdots+a_{2n}x_n=y_2$

$\cdots\cdots\cdots\cdots$

$a_{n1}x_1+a_{n2}x_2+\cdots+a_{nn}x_n=y_n$

上記方程式を、Ｂｙ＝ｘという形に変形すると、ＢがＡの逆行列となって、逆行列が求まることになる。

１）１行目の方程式をa_{11}で割る。

２）ｉ行目（ｉ＞１）－１行目×a_{i1}を演算する。

３）２行目の方程式のx_2の係数を１にするよう、２行目にx_2の係数の逆数をかける。

４）ｉ行目（ｉ＞２）－２行目×a_{i2}を演算する。

５）以上の演算をｎ－１行目まで続ける。

【０００４】

ピボットの入れ替えに伴う列ベクトルの入れ替えは以下の通りである。

Ａｘ＝ｙの両辺にピボットの入れ替え処理に対応する行列Ｐをかける。

ＰＡｘ＝Ｐｙ＝ｚ

行列Ｂについて、以下が成り立つとすると、

ｘ＝Ｂｚ

Ｂは、以下のように与えられる。

$B=(PA)^{-1}=A^{-1}P^{-1}$

つまり、求まったＢに右からＰをかけることにより、Ａの逆行列が求まる。実際には、列ベクトルの入れ替えを行う必要がある。

【０００５】

なお、ここで、$P=P_nP_{n-1}\cdots P_1$であり、P_nは、行列の要素が$P_{ii}=0$、$P_{ij}=1$、$P_{jj}=0$、$P_{ji}=1$となる直交変換である。

【０００６】

【発明が解決しようとする課題】

ベクトル計算機においては、メモリアクセス系の動作速度が速いことを前提として、上記のような方法で逆行列の演算を行っていたが、共有メモリ型スカラ計算機の場合、演算する行列が大きくなるほどに、共有メモリにアクセスする回数が多くなり、アクセス速度の遅い共有メモリへのアクセスによって、計算機の性能が大きく損なわれてしまうという問題がある。そこで、共有メモリ型スカラ計算機の各プロセッサに設けられる、アクセス速度の速いキャッシュメモリを有効に使い上記のような行列計算を行う必要がある。つまり、行列の各行あるいは列毎に演算していると、共有メモリへのアクセスが多くなってしまうので、行列をブロック化して、各プロセッサにキャッシュメモリに格納されたデータを最大限処理した後、共有メモリにアクセスするようにして、共有メモリへのアクセス数を減少する、各プロセッサに局所化したアルゴリズムが必要となる。

【０００７】

本発明の課題は、共有メモリ型スカラ計算機において、高速に逆行列の演算が行える演算の並列処理方法を提供することである。

【０００８】

【課題を解決するための手段】

本発明の方法は、共有メモリ型スカラ並列計算機用逆行列の並列処理方法であって、逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にＬＵ分解するステップと、左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、所定の正方形ブロックの更新を複数段に分けて、１つのプロセッサで行うステップと、該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと

を備えることを特徴とする。

【０００９】

本発明によれば、逆行列の演算をブロック毎の更新処理とし、各ブロックの更新を複数のプロセッサで並列に行うことにより、プロセッサに対応して設けられるキャッシュに格納されたブロックに対し、最大限演算を行った後に、共有メモリへのアクセスをするようにしているので、アルゴリズムが局所化し、高速な逆行列を求めるための演算が行える。

【００１０】

【発明の実施の形態】

本発明の実施形態では、ある行列の逆行列を求めるアルゴリズムを提供する。共有メモリ型スカラ並列計算機では、ロードに対する演算量を増やす必要がある。このため、本発明の実施形態では、行列積の形で演算を効率的に行うブロック化した方法を提供する。更に、行列積を使う更新で必要となる行列部分を大きさの変化する行列積を利用する再帰プログラムで記述することで演算密度を高める。

【００１１】

図１は、本発明の実施形態が前提とする共有メモリ型スカラ計算機のハードウェア構成を示した図である。

プロセッサ１０－１～１０－ｎは、１次キャッシュメモリを持っており、この１次キャッシュメモリはプロセッサの中に組み込まれていることもある。また、各プロセッサ１０－１～１０－ｎには、２次キャッシュメモリ１３－１～１３－ｎが設けられ、２次キャッシュメモリ１３－１～１３－ｎが相互結合網１２に結合されている。また、相互結合網１２には、共有メモリであるメモリモジュール１１－１～１１－ｎが設けられ、プロセッサ１０－１～１０－ｎは、演算に必要なデータをここから読み出し、相互結合網１２を介して、２次キャッシュメモリ１３－１～１３－ｎあるいは、１次キャッシュメモリに記憶させて、演算を行う。

【００１２】

この場合、メモリモジュール１１－１～１１－ｎから２次キャッシュメモリ１

３－１～１３－ｎあるいは１次キャッシュメモリにデータを読み込んだり、２次キャッシュメモリ１３－１～１３－ｎ、あるいは、１次キャッシュメモリから演算後のデータをメモリモジュール１１－１～１１－ｎに書き込むのはプロセッサ１０－１～１０－ｎの演算速度に比べて非常に遅い。従って、このような書き込み、読み出しが頻繁に発生すると計算機全体の性能を劣化させてしまう。

【００１３】

従って、計算機全体の性能を高く維持するためには、メモリモジュール１１－１～１１－ｎへのアクセスをできるだけ減らし、２次キャッシュメモリ１３－１～１３－ｎ、１次キャッシュメモリ、及びプロセッサ１０－１～１０－ｎからなるローカルな系においてできるだけ多くの演算をしてしまうようなアルゴリズムが必要となる。

【００１４】

したがって、本願発明の実施形態においては、以下のようにして逆行列の計算を行う。

演算すべき行列内のあるブロック幅のブロックに関して右方向の更新のみを行う。こうすることで、ブロックの外での更新で使う情報を残すようにする。つまり、消去の過程で本来機軸として選んだ列ベクトルと直交する行ベクトルの外積で残りの部分を更新することが従来の技術で説明したＧａｕｓｓ－Ｊｏｒｄａｎの方法において行われることが分かる。すなわち、前述のＧａｕｓｓ－Ｊｏｒｄａｎの方法の２）の方法をあらわに書くと、

$a_{ij} - c_{1j} \times a_{i1}$となっている（ｉ、ｊ＞１：$c_{1j} = a_{1j} / a_{11}$）。

【００１５】

この式の意味するところは、１列目の列ベクトルを機軸として選んで、これと直交する１行目の行ベクトルを選択すると、これらを除いた２行目以降×２列目以降からなる行列は、１列目の列ベクトルと１行目の行ベクトルの外積で更新されるということである。

【００１６】

また、Ａｘ＝ｙをＩを単位行列として書き直すと、Ａｘ＝Ｉｙとなり、ＡをＧａｕｓｓ－Ｊｏｒｄａｎ法によって変換した場合、右辺の単位行列も変換を受け

る。

【００１７】

この単位行列の更新においては、Ａにおいて消去された列に対応する列の（ｉ、ｉ）要素は、右辺の行列においては１／a_{ii}となり、右辺の行列の他の要素は、Ａの消去された列ベクトルの要素に１／a_{ii}をかけて－１を乗じた数となる。また、列ブロック行列でも左側の消去が必要になる。

【００１８】

図２～図８は、本発明の実施形態の計算順序を説明する図である。

計算の順序は、次のようになる。

まず、図２において、行列Ｍを列ブロック行列毎に左からブロック化して、上で述べた方法で更新を行う。

１）ＥとＨをＬＵ分解する（特願平１２―３５８２３２号参照）。

２）ＢをＥの上三角部分Ｕを利用してＢ←BU^{-1}と更新する。

３）Ｅの下三角部分を利用してＤ←$L^{-1}D$及びＦ←$L^{-1}D$

４）Ａ、Ｃ、Ｇ、Ｉを更新する。

【００１９】

Ａ←Ａ－Ｂ×Ｄ、Ｃ←Ｃ－Ｂ×Ｆ、Ｇ←Ｇ－Ｈ×Ｄ、Ｉ←Ｉ－Ｈ×Ｆ

５）Ｅの上三角部分の更新を行う。この部分はＬＵ分解では未更新となっているため、このタイミングで更新する。

６）次にＤ及びＦの上部に対する更新をＤ、Ｆの情報とＥの情報を使って更新する。

【００２０】

このとき、演算密度を高めるために再帰的なプログラムで行列積を使って更新する。

ａ）再帰的プログラム

Ｄ及びＦの上部の更新は、各々の行ベクトルとＥのそれと直交する列ベクトルの外積で計算できる。演算密度を上げるために次のような再帰的プログラムで行う。ここで、Ｄ及びＦの上部とは、図８で説明するように、Ｄ及びＦの行ブロックの上の方と言う意味であり、図８から明らかなように、Ｄ及びＦの更新は、こ

れら行ブロックの上部から行われる。

【００２１】

recursive　subroutine　rowupdate ()

if　（更新幅＜１０）then

Ｅの上三角行列から対角要素を除いた行列と行ブロック行列をかけて差し引く（詳細は、図８の説明参照）

else

c　更新幅を前半と後半に分ける

前半部分を更新する

c　次に後半部分をrowupdateを呼び出して更新する

call　rowupdate ()

return

end

７）次に対角線上の要素を右下に移動させながら、この対角の左下の列方向の長方形部分を更新する。この結果、Ｂ及びＨの更新に必要な情報を作り出す。

８）Ｂ、Ｈに関して左側の更新を行う。これは以下のように行う。

【００２２】

ブロック幅をｄとして、左からｉ、・・・、ｉ＋ｄ－１と番号を振る。

a_{ii}を逆数にする。他の部分はa_{ii}で割って、符号を逆にする。

ｉ＋１行の左を$a_{i+1,i+1}$で割り、$a_{i+1,i+1}$を逆数にする。

【００２３】

左部分の更新を左部分のｉ＋１行とｉ＋１列をかけて引くことで行う。

これを繰り返す。

この部分は、更に以下の手順に分かれる。これらも再帰的なアルゴリズムに組み立て直すことが必要となる。

９）最後にＥの対角線上の要素から見て左上部分の行方向の長方形部分を更新する。

１０）最後に列ベクトルをピボットの入れ替えの履歴の逆方向に入れ替える処理を行う。

５）で更新する部分は図３の斜線部分（Ｅ２）を、対角要素を対角線にスライドさせながら

Ｅ２＝Ｅ２－ａ×ｂで更新する。

【００２４】

この部分はＬＵ分解の処理では未更新である。これを行うことで、Ｄ、Ｆの更新で必要になる上三角行列の情報が得られる。

７）で更新する部分は図４の斜線部分を対角要素をスライドさせながら、Ｅ３＝Ｅ３－ｃ×ｄで更新する。

【００２５】

この更新に先立ち、ｃは対角要素の逆数をかける。

更新の後の対角要素は元の対角要素の逆数とする。ｄは、各列要素に更新前の対角要素をかけて、－１をかけたものとする。

【００２６】

この結果、Ｂ及びＨの左方向の更新で必要な下三角行列の情報が得られる。

９）で、図５の斜線の部分を対角要素をスライドさせながら、

Ｅ１＝Ｅ１－ａ×ｃで更新する。

【００２７】

更新の後のａは、ａの列要素に元の対角要素をかけて、－１をかけたものとする。

Ｄ）共有メモリ型スカラ並列計算機向けの並列化の詳細

０）Ｅ、Ｆ、Ｈ及びＩのＬＵ分解はＬＵ分解の並列化アルゴリズムを利用して並列分解する。

１）ブロック幅の制御

本発明の実施形態では、問題規模及び並列化で使うプロセッサ数に応じて、ブロック幅を調節する機構を備える。

【００２８】

ブロックＥの更新は一つのプロセッサで行うので、この部分のコストが全体のコストから考えて無視できる程度（１％程度）になるようにブロック幅を決める。

２）Ｃの４）の更新はおのおのの行列積による更新を並列に行う。各プロセッサで２次元目を均等に分割して計算を並列分担する。

３）Ｃの５）での更新、つまりＤ及びＦの更新はＥの上三角部分を共通に参照して、Ｄ及びＦの２次元目を均等に分割した領域を各プロセッサが並列に更新する。

４）Ｃの８）の更新は及びＨの１次元目を均等に分割した領域を各プロセッサは並列分担して更新する。

５）最後に、列ベクトルの入れ替え処理は、行列全体の１次元目を均等に分割して、各領域の部分に関する入れ替えを並列に行う。

【００２９】

図６の破線は各行列部分を並列に計算する更新を並列に行う領域の分割例を示している。

・ＢまたはＨの更新についてのメモリのアクセス

深さ２まで再帰プログラムが動作した場合を解説する。

【００３０】

図７（ａ）と（ｂ）、（ｃ）と（ｄ）、（ｅ）と（ｆ）が組になっている。

最初に、図７（ａ）の斜線部分を更新する。このとき斜線部分及び図７（ｂ）の破線三角部分を使う。

【００３１】

詳細は、疑似コード参照。次に、図７（ａ）斜線部分を図７（ａ）横線部分と図７（ｂ）斜線四角部分を使って更新する。その後、図７（ａ）横線部分を図７（ｂ）太線三角部分を使って更新する。

【００３２】

次に、図７（ｃ）斜線部分を、図７（ｃ）右白塗り四角と図７（ｄ）太線四角の積で更新する。

その後、図７（ｅ）斜線部分を図７（ｆ）破線三角部分を使い更新する。更に、図７（ｅ）斜線部分を図７（ｅ）横線部分と図７（ｆ）斜線四角の積で更新する。最後に横線部分を図７（ｆ）太線三角部分を使って更新する。

【００３３】

上記手順において、Ｅの参照は共通である。従って、Ｅは各プロセッサのキャッシュに格納して、参照するようにしても良い。また、例えば、Ｂの参照・更新は行方向（太線破線で分割）した領域に関して並列に処理する。

【００３４】

・ＤまたはＦの更新におけるメモリアクセス

深さ２まで再帰プログラムが動作した場合を解説する。

【００３５】

最初に、図８（ａ）左の斜線部分を更新する。このとき斜線部分及び図８（ａ）右の破線三角部分を使う。

詳細は、疑似コード参照。次に、図８（ａ）左の斜線部分を図８（ａ）右の斜線四角部分と図８（ａ）左の縦線部分を使って更新する。その後、図８（ａ）左の横線部分を図８（ａ）右の太線三角分を使って更新する。

次に、図８（ｂ）左の斜線部分を、図８（ｂ）右の太線四角と図８（ｂ）左の白塗り四角の積で更新する。

その後、図８（ｃ）左の斜線部分を図８（ｃ）右の破線三角部分を使い更新する。更に、図８（ｃ）左の斜線部分を図８（ｃ）右の斜線四角と図８（ｃ）左の縦線部分との積で更新する。最後に図８（ｃ）左の縦線部分を図８（ｃ）右の太線三角部分を使って更新。

【００３６】

上記手順において、Ｅ参照は共通である。従って、各プロセッサのキャッシュにＥを格納して参照するようにしても良い。また、Ｄの参照・更新は行方向（太線破線で分割）した領域に関して並列に処理する。

【００３７】

図９～図１５は、本発明の実施形態の疑似コードである。

図９は、逆行列の並列化アルゴリズムのメインアルゴリズムの疑似コードである。以下の疑似コードで、左端にｃと付いている行はコメント行である。array a(k,n)は、逆行列を求めるべき行列の要素を格納する配列である。ip（n）は、ＬＵ分解のサブルーチンにおいて、行の入れ替えを行った際の情報が格納される配列である。ＬＵ分解のサブルーチンのアルゴリズムは、特願平１２－３５８２３２号を参照されたい。ｎｂは、ＬＵ分解する際に指定するブロックの数を示す。

【００３８】

１つの指定ブロックについてのＬＵ分解が終わると、ｉｐ（ｉ）がｉより大きいとき、行列のｉ行目をｉｐ（ｉ）行目と入れ替える。そして、ｕｐｄａｔｅサブルーチンを呼び出して、行列の更新を行う。ＬＵ分解からｕｐｄａｔｅサブルーチンまでの処理は、指定されるブロック全てについて処理し終わるまで繰り返し行う。そして、最後の指定ブロックについては、別途ＬＵ分解とｕｐｄａｔｅをおこなって処理を終了する。

【００３９】

図１０は、ブロックＬＵ分解の情報を利用して残りの部分の更新を行うルーチンの疑似コードである。

同図の更新ルーチンでは、ブロックＡ～Ｈに従って、それぞれの更新を行う。ブロックＡ～Ｄ及びＧについては、専用のサブルーチンを更に読み出す。ブロックＩは、ＬＵ分解の際に既に更新されているので、ここでは別途サブルーチンを設けることはしない。

【００４０】

ブロックＡ～Ｄ及びＧの更新ルーチンが終了した後、バリア同期を取る。その次に、プロセッサの番号（スレッドの番号）が１の場合に、「ｅの更新１」を行う。これは、ｅ－ｕｐｄａｔｅ１サブルーチンによって行う。その後、バリア同期を取る。

【００４１】

ｌｅｎは、１つのスレッドが処理するブロックの幅である。ｉｓは、処理するブロックの最初の位置であり、ｉｅは、処理するブロックの最後の位置である。

ｄｆ－ｕｐｄａｔｅは、ブロックＤ及びＦを更新するサブルーチンである。ブロックＤ、Ｆの更新が終わると、ブロックの最初の位置にブロック幅を加えたものを新たなブロックの最初の位置（nbase2）として格納し、ｌｅｎを新たに計算し、ブロックの最初と最後の位置ｉｓ２とｉｅ２を新たに計算し、ｄｆ－ｕｐｄａｔｅでＤとＦの更新を行い、バリア同期を取る。

【００４２】

更に、「ｅの更新２」として、スレッドの番号が１の場合に、ブロックＥの更新サブルーチンｅ－ｕｐｄａｔｅ２を呼び出し、バリア同期を取る。上記と同様に、ｌｅｎ、ｉｓ、ｉｅを計算し、ブロックＢとＨの更新ルーチンｂｈ－ｕｐｄａｔｅを呼び出し、その後、nbase2を求めて、ｌｅｎ、ｉｓ２、ｉｅ２を求め、再びｂｈ－ｕｐｄａｔｅで処理を行い、バリア同期を取る。

【００４３】

更に、スレッドの番号が１の時、「ｅの更新３」として、ｅ－ｕｐｄａｔｅ３で処理を行い、処理後バリア同期を取る。

更に、その後、ピボットの入れ替えたままの状態を元に戻すため、ｌｅｎ、ｉｓ、ｉｅを計算した後、サブルーチンｅｘｃｈａｎｇｅによって列の入れ替えを行って、バリア同期を取り、スレッドを消去して、処理を終了する。

【００４４】

図１１は、ブロックＢとブロックＤの更新サブルーチンの疑似コードである。

ブロックＢの更新においては、サブルーチンｂ－ｕｐｄａｔｅは、共有の行列配列a(k,n)にアクセスし、上記説明と同じ意味を持つｌｅｎ、ｉｓ１、ｉｅ１を計算する。ｉｏｆは、Ｂブロックの始まりの列の番号である。そして、Ｅブロックの上三角行列の対角要素を１にした行列ＴＲＵ－Ｕを使って、同図に示す式によって行列ａのブロックＢの更新をする。ここで、ｉｓ：ｉｅという記号は、行列要素のｉｓからｉｅまでについて処理をすることを意味する。

【００４５】

また、ブロックＤの更新においては、サブルーチンｄ－ｕｐｄａｔｅによって、同様のパラメータを計算し、ブロックＥの下三角行列ＴＲＬによって同図の式のように行列ａの更新を行う。

【００４６】

図１２は、ブロックＣとＡの更新サブルーチンの疑似コードである。

ブロックＣの更新サブルーチンｃ－ｕｐｄａｔｅでは、ブロックＢとＦの乗算によってブロックＣを更新する。ａ（１：ｉｏｆ、ｉｓ２：ｉｅ２）がブロックＣを表し、ａ（１：ｉｏｆ、ｉｏｆ＋１：ｉｏｆ＋ｂｌｋ）がブロックＢを、ａ（ｉｏｆ＋１：ｉｏｆ＋ｂｌｋ、ｉｓ２：ｉｅ２）がブロックＦを表す。

【００４７】

ブロックＡの更新サブルーチンａ－ｐｕｄａｔｅでは、ブロックＢとＤでブロックＡを更新する。ａ（１：ｉｏｆ、ｉｓ２：ｉｅ２）がブロックＡ、ａ（１：ｉｏｆ、ｉｏｆ＋１：ｉｏｆ＋ｂｌｋ）がブロックＢ、ａ（ｉｏｆ＋１：ｉｏｆ＋ｂｌｋ、ｉｓ２：ｉｅ２）がブロックＤである。

【００４８】

図１３は、ブロックＧ、Ｅの最初と２回目の更新を表す疑似コードである。

ブロックＧの更新サブルーチンｇ－ｕｐｄａｔｅにおいては、前述のサブルーチンと同様に、ブロックの幅や開始位置、終了位置などを示すｌｅｎ、ｉｓ２、ｉｅ２、ｉｏｆなどを計算し、ブロックＧをブロックＤとＨで更新する。ａ（ｉｏｆ＋１：ｎ、ｉｓ２：ｉｅ２）がブロックＧであり、ａ（ｉｏｆ＋１：ｎ、ｉｏｆ＋１：ｉｏｆ＋ｂｌｋ）がブロックＨ、ａ（ｉｏｆ＋１：ｉｏｆ＋ｂｌｋ、ｉｓ２：ｉｅ２）がブロックＤである。

【００４９】

ブロックＥの最初の更新サブルーチンｅ－ｕｐｄａｔｅ１では、Ｅの対角成分より上の三角行列を対角成分以前の列ベクトルｓ（１：ｉ、ｉ）と、該対角成分以降の行ベクトル（ｉ、ｉ＋１：ｂｌｋ）で更新する。

【００５０】

ブロックＥの２回目の更新サブルーチンｅ－ｕｐｄａｔｅ２では、ブロックＥの上三角行列の対角成分を更新前の要素値を対角成分値で割った値に更新し、対角成分以前の行ベクトルｓ（ｉ、１：ｉ－１）と対角成分以後の列ベクトルｓ（ｉ＋１：ｂｌｋ，ｉ）で更新し、ブロックＥの下三角行列の要素値を対角成分の符号を変えたもので割った値に更新し、対角要素にブロックＥの対角要素の逆数

に更新する。

【００５１】

図１４は、ブロックＥの最後の更新、ブロックＤ及びＦの更新サブルーチンの疑似コードである。

ブロックＥの最後の更新サブルーチンｅ－ｕｐｄａｔｅ３では、ブロックＥの上三角行列を対角要素以前の列ベクトルｓ（１：ｉ－１、ｉ）と行ベクトルｓ（ｉ、１：ｉ－１）で更新し、ブロックＥの対角要素以前の要素に更新前の対角要素をかけて更新する。

【００５２】

ブロックＤ及びＦの更新サブルーチンｄｆ－ｕｐｄａｔｅにおいては、ブロックの幅ｌｅｎが１０より小さい場合、ブロックＤあるいはブロックＦ（サブルーチンの引数のｉｓ、ｉｅによって決定される）をブロックＥの要素値ｓ（１：ｉ－１、ｉ）と自身の行ベクトルａ（ｉ、ｉｓ：ｉｅ）で更新する。ここで、ブロックＤあるいはＦの要素値がａ（１：ｉ－１、ｉｓ：ｉｅ）となっているのは、このサブルーチンが行列要素を読み込む場合、前述のnbaseによって読み込む位置がオフセットされて読み込まれているため、列番号が１～ｉ－１の要素値について計算することがブロックＤあるいはＦについて研鑽することになる。ｌｅｎが２０以上、３２以下の場合には、ｌｅｎ１、ｌｅｎ２を定義し、ｄｆ－ｕｐｄａｔｅを再帰的に呼び出し、同図の処理をした後、更に、ｄｆ－ｕｐｄａｔｅを呼び出して処理を行い終了する。

【００５３】

図１５は、ブロックＢ及びＨの更新サブルーチンの疑似コードである。

同図においては、ｂｈ－ｕｐｄａｔｅは、ｌｅｎが１０より小さいとき、同図の演算によって更新し、ｌｅｎが２０以上、３２以下の時には、ｌｅｎ１、ｌｅｎ２を定義し、その他の場合には、ｌｅｎ１、ｌｅｎ２を別に定義し、ｂｈ－ｕｐｄａｔｅを呼び出し、同図の式による演算を行い、更にｂｈ－ｕｐｄａｔｅを呼び出し、処理をして終了する。

【００５４】

本発明の実施形態によれば、同じ機能（ＬＵ分解した後、逆行列を求める別の

方法）で、ＳＵＮの数値計算ライブラリSUN　Performance　libraryの機能に比べて７個のＣＰＵで６．６倍高速となる。

【００５５】

図１６～図２９は、疑似コードの処理をフローチャートで表したものである。

図１６は、メインの処理である。まず、逆行列の演算開始として、サブルーチンとしてshared配列Ａ（ｋ、ｎ）を入力する。ステップＳ１０では、スレッドを生成し、各スレッドでローカル域numthrに総スレッド数、nothrdに各スレッドに割り振られたスレッド番号を設定する。また、各スレッドでiblkにブロック幅を設定し、nb=（n+iblk-1)/iblkを設定し、i=1を設定する。ステップＳ１１では、iがnb-1に等しいか否かを判断する。ステップＳ１１の判断がＹＥＳの場合には、ステップＳ１７に進む。ステップＳ１１の判断がＮＯの場合には、ステップＳ１２において、nbaseを(i-1)×iblkを設定し、ステップＳ１３で、Ａ（nbase+1:n、nbase+1:nbase+n）の部分をブロック幅iblkでブロックＬＵ分解し、行ブロック及び右下正方行列部分を更新する。ここで、ip（nbase+1:nbase+iblk）に行の入れ替えの情報が入っている。これはサブルーチンにしておく。この部分の並列アルゴリズムは先に本出願人が出願した特許出願に記載されている。ステップＳ１４においては、サブルーチンexchgrowを呼び出し、Ａ（nbase+1:nbase+iblk、１：nbase）部分の行ベクトルの入れ替えを行う。すなわち、ｊをnbase+1からnbase+iblkまで変化させて、ｉｐ（ｊ）＞ｊを満たすときＡ（j,1:nbase）とＡ（ip(j),1:nbase）を入れ替える。ステップＳ１５では、サブルーチンupdateを呼んで、他のブロックを更新する。ステップＳ１６では、i=i+1として、ステップＳ１１に戻る。ステップＳ１７では、nbase=(nb-1)×iblksを演算し、ステップＳ１８において、Ａ（nbase+1:n,nbase+1:n）の部分に関して、ＬＵ分解を行う。ip（nbase+1:n-1）に行の入れ替えの情報が入っている。ステップＳ１９において、サブルーチンexchgrowを呼び出し、Ａ（nbase+1:n-1,1:nbase）部分の行ベクトルの入れ替えを行う。ステップＳ２０において、サブルーチンupdateを呼んで更新する。ステップＳ２１では、サブルーチンexchgcolを呼んで列ベクトルを入れ替える。すなわち、ｊをｎから１づつ減らして変化させて、ip(j)＞jを満たすとき、Ａ（1:n,j）、Ａ（1:n,ip(j)）を入れ替える。ステップＳ２２では、並列

処理のために生成したスレッドを消去する。

【００５６】

図１７及び図１８は、サブルーチンupdateのフローチャートである。

ステップＳ３０においては、サブルーチンb-update、d-update、c-update、a-update、g-updateを呼び出すとき、nbase、iblk、配列Ａ及びスレッドの情報であるnumthrd、各スレッドの番号nothrdを受け渡す。サブルーチンb-updateを呼んでブロックを更新する。ステップＳ３１では、サブルーチンd-updateを呼んでブロックを更新する。ステップＳ３２において、サブルーチンc-updateを呼んでブロックを更新する。ステップＳ３３では、サブルーチンa-updateを呼んでブロックを更新する。ステップＳ３４では、サブルーチンg-updateを呼んでブロックを更新する。

【００５７】

ステップＳ３５においては、各スレッド間でバリア同期を取る。ステップＳ３６においては、nothrdの値が１のスレッドか否かを判断する。ステップＳ３６の判断がＮＯの場合には、ステップＳ３８に進む。ステップＳ３６の判断がＹＥＳの場合には、ステップＳ３７において、サブルーチンe-updateを呼んでブロックｅを更新し、ステップＳ３８に進む。ステップＳ３８では、各スレッド間でバリア同期を取る。ステップＳ３９においては、df-updateで各スレッドで分担する始点（is）、終点（ie）を決めて、受け渡す。len=(nbase+numthrd-1)/numthrd、is=(nothrd-1)×len+1, ie=min(nbase, nothrd×len) を演算する。ブロックの１次元目先頭istart=nbase+1、ブロック幅len=iblkとする。ステップＳ４０においては、サブルーチンdf-updateを呼んでブロックｆを更新する。

【００５８】

ステップＳ４１においては、同じく始点、終点を計算し受け渡す。nbase2=nbase+iblk, len=(n-nbase2+numthrd-1)/numthrd、is2=nbase2+(nothrd-1)×len+1、ie2=min(n, nbase2+nothrd×len) を演算する。ステップＳ４２においては、ブロックの１次元目先頭istart=nbase+1、ブロック幅len=iblkとする。サブルーチンdf-updateを呼んで、ブロックｆを更新する。ステップＳ４３では、各スレッド間でバリア同期をとる。ステップＳ４４nothrdの値が１のスレッドか否かを判断

する。ステップＳ４４の判断がＮＯの場合には、ステップＳ４６に進む。ステップＳ４４の判断がＹＥＳの場合には、ステップＳ４５において、サブルーチンe-update2を呼んでブロックｅを更新し、ステップＳ４６に進む。ステップＳ４６においては、各スレッド間でバリア同期を取る。ステップＳ４７においては、bh-updateの各スレッドに受け渡す始点、終点を計算する。すなわち、len=(nbase+numthrd-1)/numthed、is=(nothrd-1)×len+1、ie=min(nbase,nothrd×len) を計算する。ブロックの１次元目の先頭istart=nbase+1、ブロック幅len=iblkとする。サブルーチンbh-updateを呼んで、ブロックｂを更新する。ステップＳ４８においては、bh-updateの各スレッドに受け渡す始点、終点を計算する。

【００５９】

ステップＳ４９にいては、nbase2=nbase+iblk、len=(n-nbase2+numthrd-1)/numthrd、is2=nbase2+(nothrd-1)×len+1、ie2=min(n,nbase2+nothrd×nothrd×len)を計算し、ブロックの１次元目の先頭istart=nbase+1、ブロック幅len=iblkとする。ステップＳ５０においては、bh-updateの各スレッドに受け渡す始点、終点を計算する。ステップＳ５１においては、各スレッド間でバリア同期をとる。ステップＳ５２では、nothrdの値が１のスレッドか否かを判断する。ステップＳ５２の判断がＮＯの場合には、ステップＳ５４に進み、判断がＹＥＳの場合には、ステップＳ５３において、サブルーチンe-update3を呼んで、ブロックｅを更新し、ステップＳ５４に進む。ステップＳ５４においては、各スレッド間でバリア同期を取る。

【００６０】

図１９は、サブルーチンb-updateとd-updateの処理を示すフローチャートである。

サブルーチンb-updateでは、ステップＳ６０において、各スレッドで分担する１次元目の始点、終点を計算する。すなわち、len=(nbase+numthrd-1)/numthrd、isl=(nothrd-1)×len+1、iel=min(nbase,nothrd×len) 、iof=nbaseを計算する。ステップＳ６１においては、A (isl:iel,iof+1:iof+iblk) ＝A (isl:iel,iof+1:iof+iblk)×TRU-U (A(iof+1:iof+iblk,iof+1:iof+iblk)) $^{-1}$を計算する。TRU-Uは対角要素を１．０とした上三角行列部分である。そして、サブルーチ

ンを抜ける。

【００６１】

サブルーチンd-updateでは、ステップＳ６５において、各スレッドで分担する始点、終点を計算する。すなわち、len=(nbase+numthrd-1)/numthrd、isl=(nothrd-1)×len+1、iel=min(nbase,nothrd×len)、iof=nbaseを計算する。ステップＳ６６では、A(iof+1:iof+iblk,isl:iel)=TRL(A(iof+1:iof+iblk,iof+1:iof+iblk))$^{-1}$×A(iof+1:iof+iblk,isl:iel)を計算する。TRLは正方行列の下三角行列部分である。そして、サブルーチンを抜ける。

【００６２】

図２０は、サブルーチンc-updateのフローチャートである。

ステップＳ７０において、各スレッドで分担する２次元目の始点、終点を計算する。すなわち、nbase2=nbase+iblk、len=(n-nbase2+numthrd-1)/numthrd、is2=nbase2+(nothrd-1)×len+1、ie2=min(n,nbase2+nothrd×len)、iof=nbaseを計算する。ステップＳ７１では、A(1:iof、isl:ie2)=A(1:iof,iof+1:iof+ilbk)×A(iof+1:iof+ilbk,is2:ie2)を計算し、サブルーチンを抜ける。

【００６３】

図２１は、サブルーチンa-updateのフローチャートである。

ステップＳ７５において、各スレッドで分担する２次元目の始点、終点を計算する。すなわち、len=(nbase+numthrd-1)/numthrd、is2=(nothrd-1)×len+1、ie2=min(nbase,nothrd×len)、iof=nbaseを計算する。ステップＳ７６においては、A(1:iof,is2:ie2)=A(1:iof,is2:ie2)-A(1:iof,iof+1:iof+iblk)×A(iof+1:iof+iblk,is2:ie2)を計算して、サブルーチンを抜ける。

【００６４】

図２２は、サブルーチンg-updateのフローチャートである。

ステップＳ８０においては、各スレッドで分担する２次元目の始点、終点を計算する。すなわち、len=(nbase+numthrd-1)/umthrd、is2=(nothrd-1)×len+1、ie2=(nbase,nothrd×len)、iof=nbaseを計算する。ステップＳ８１においては、A(iof+1:n,is2:ie2)＝A(iof+1:n、is2:ie2)-A(iof+1:n,iof+1:iof+iblk)×A(iof+1:iof+iblk,is2:ie2)を計算し、サブルーチンを抜ける。

【００６５】

図２３は、サブルーチンe-updateのフローチャートである。

このルーチンでは、ブロックＥの左上隅の要素がＳ（ｋ、＊）の最初の要素となるように引数で受け取る。ステップＳ８５では、i=1を設定する。ステップＳ８６では、i>iblkか否かを判断する。ステップＳ８６の判断がＹＥＳの場合には、サブルーチンを抜ける。ステップＳ８６の判断がＮＯの場合には、ステップＳ８７において、S(1:i-1, i+1:iblk) =S (1:i-1, i+1:iblk) -S (1:i-1, i) ×S (i, i+1:iblk) 、i=i+1を計算し、サブルーチンを抜ける。

【００６６】

図２４は、サブルーチンe2-updateのフローチャートである。

まず、ブロックＥの左上隅の要素がＳ（ｋ、＊）の最初の要素となるように引数で受ける。ステップＳ９０では、i=1と設定し、ステップＳ９１において、i>iblkか否かを判断する。ステップＳ９１の判断がＹＥＳの場合には、サブルーチンを抜ける。ステップＳ９１の判断がＮＯの場合には、ステップＳ９２において、tmp=1.0/S(i, i)、S(i, i:i-1)=tmp×S(i, 1:i-1)、S(i+1:iblk, 1:i-1)=S(i+1:iblk, 1:i-1)-S(i, 1:i-1)×S(i+1:iblk, i)、S(i+1:iblk, i)=-A (i+1, iblk, i)×tmp、A (i, i) =tmp、i=i+1を演算して、ステップＳ９１に戻る。

【００６７】

図２５は、サブルーチンe3-updateのフローチャートである。

まず、ブロックＥの左上隅の要素がＳ（ｋ、＊）の最初の要素となるように引数で受ける。ステップＳ９５では、i=1と設定し、ステップＳ９６で、i>iblkか否かを判断する。ステップＳ９６の判断がＹＥＳの場合には、サブルーチンを抜ける。ステップＳ９６の判断がＮＯの場合には、ステップＳ９７において、S(1:i-1, 1:i-1)=S(1:i-1, 1:i-1)-S(1:i-1, i)×S(i, 1:i-1)、S(1:i-1, i)=S(1:i-1)×S(i, i)、i=i+1を演算し、ステップＳ９６に戻る。

【００６８】

図２６は、サブルーチンdf-updateのフローチャートである。

このサブルーチンは、再帰的プログラムとなっている。まず、各スレッドで処理する範囲を示す始点、終点をis、ieで受ける。ブロックＥの左上隅の要素がＳ

（ｋ、＊）の最初の要素となるように引数で受ける。処理するブロックの１次元目の先頭istartとブロック幅lenで受ける。ステップＳ１００では、len<10か否かを判断する。ステップＳ１００の判断がＮＯの場合には、ステップＳ１０４に進む。ステップＳ１００の判断がＹＥＳの場合には、ステップＳ１０１において、i=1と設定し、ステップＳ１０２において、i<=lenか否かを判断する。ステップＳ１０２の判断がＮＯの場合には、サブルーチンを抜ける。ステップＳ１０２の判断がＹＥＳの場合には、ステップＳ１０３において、js=istart、je=istart-1+len-1、Ａ（js:je,is,ie)=Ａ（js:je,is,ie)-Ｓ（js:je,i）×Ａ（istart+i-1,is:ie）を演算し、ステップＳ１０２に戻る。

【００６９】

ステップＳ１０４では、len>=32あるいはlen<=20か否かを判断する。ステップＳ１０４の判断がＮＯの場合には、ステップＳ１０６で、len1=len/3、len2=len-len１として、ステップＳ１０７に進む。ステップＳ１０４の判断がＹＥＳの場合には、ステップＳ１０５において、len1=len/2、len2=len-len1として、ステップＳ１０７に進む。ステップＳ１０７では、サブルーチンbf-updateを再帰的に呼び出す（ブロックの先頭のistartとブロック幅のlen１を受け渡す）。ステップＳ１０８では、js=istart、je=istart+len1-1、js2=js-nbase、je2=js2+len1-1、js3=js2+len1、je3=js3+len2-1、js4=istart+len1、je4=js4+len2-1、Ａ（js:je,is:ie）=Ａ（js:je,is:ie）-Ｓ（js2:je2,js3:je3）×Ａ（js4,je4:len,is:ie）、istart2=istart+len1を演算して、ステップＳ１０９に進む。ステップＳ１０９では、サブルーチンdf-updateを再帰的に呼び出し（ブロックの先頭のistart2とブロック幅のlen2を受け渡す）、サブルーチンを終了する。

【００７０】

図２７は、サブルーチンbh-updateのフローチャートである。

このサブルーチンは再帰的プログラムである。まず、各スレッドで処理する範囲を示す始点及び終点をis、ieで受ける。また、ブロックＥの左上隅の要素がＳ（ｋ、＊）の税所の要素となるように引数で受ける。処理するブロックの先頭istartとブロック幅をlenを受ける。

【００７１】

ステップＳ１１５では、len<10か否かを判断する。ステップＳ１１５の判断がＮＯの場合には、ステップＳ１１９に進む。ステップＳ１１５の判断がＹＥＳの場合には、ステップＳ１１６において、i=1と設定し、ステップＳ１１７において、i<=lenか否かを判断する。ステップＳ１１７の判断がＮＯの場合には、サブルーチンを抜ける。ステップＳ１１７の判断がＹＥＳの場合には、ステップＳ１１８において、j=istart+i-1、j2=j-nbase、je=istart+len-1、je2=je-nbase、A（is:ie,j）=-A（is:ie,j）×S（j2,j2）、A（is:ie,j）=A（is:ie,j）-A（is:ie,j+1:je）×S（j2+1:je2,j2）を演算して、ステップＳ１１７に進む。

【００７２】

ステップＳ１１９では、len>=32あるいはlen<=20か否かを判断する。ステップＳ１１９の判断がＮＯの場合には、ステップＳ１２１に進み、判断がＹＥＳの場合には、ステップＳ１２０に進む。ステップＳ１２０においては、len1=len/2、len2=len-len1を計算し、ステップＳ１２２に進む。ステップＳ１２１においては、len1=len/3、len2=len-len1を演算し、ステップＳ１２２に進む。ステップＳ１２２においては、サブルーチンbh-updateを再帰的に呼び出し、対象ブロックの先頭istartとブロック幅len1を受け渡す。ステップＳ１２３では、js=istart、je=istart+len1-1、js2=js-nbase、je2=js2+len1-1、js3=istart+len1、je3=js2+len2-1、js4=js3-nbase、je4=js4+len2-1、A（is:ie,js:je）=A（is:ie,js:je）-A（is:ie,js3:je3）×S（js4:jse,js2:je2）、istart2=istart+len1を演算する。ステップＳ１２４においては、サブルーチンbh-updateを再帰的に呼び出し、対象ブロックの先頭istart2とブロック幅len1を受け渡して、サブルーチンを抜ける。

【００７３】

図２８は、サブルーチンexchgrowのフローチャートである。

ステップＳ１３０において、バリア同期を取る。ステップＳ１３１において、各スレッドで分担する始点、終点を計算する。すなわち、len=(nbase+numthrd-1)/numthrd、is=(nothrd-1)×len+1、ie=min(nbase,nothrd×len)、j=nbase+1を計算する。ステップＳ１３２では、j>min(n-1,nbase+iblks)であるか否かを判

断し、判断がＹＥＳの場合には、ステップＳ１３６に進み、判断がＮＯの場合には、ステップＳ１３３に進む。ステップＳ１３３では、ip(j)>jか否かを判断する。ステップＳ１３３の判断がＮＯの時は、ステップＳ１３５に進む。ステップＳ１３３の判断がＹＥＳの時は、ステップＳ１３４において、Ａ（j,is:ie）とＡ（ip(j),is:ie）を入れ替え、ステップＳ１３５に進む。ステップＳ１３５において、j=j+1を演算し、ステップＳ１３２に戻る。ステップＳ１３６では、バリア同期を取って、サブルーチンを抜ける。

【００７４】

図２９は、サブルーチンexchgcolのフローチャートである。

ステップＳ１４０において、バリア同期を取る。ステップＳ１４１において、各スレッドで分担する始点、終点を計算する。すなわち、len=(n+numthrd-1)/numthrd、is=(nothrd-1)×len+1、ie=min(nbase,nothrd×len）、j=n-1を計算し、ステップＳ１４２に進む。ステップＳ１４２では、j<1を判断する。ステップＳ１４２の判断がＹＥＳの場合には、ステップＳ１４６に進む。ステップＳ１４２の判断がＮＯの場合には、ステップＳ１４３において、jp(j)>jを判断する。ステップＳ１４３の判断がＮＯの場合には、ステップＳ１４５に進む。ステップＳ１４３の判断がＹＥＳの場合には、ステップＳ１４４において、Ａ（is:ie,j）とＡ（is:ie,jp(j)）を入れ替えて、ステップＳ１４５に進む。ステップＳ１４５では、J=J-1として、ステップＳ１４２に戻る。ステップＳ１４６では、バリア同期を取って、サブルーチンを抜ける。

行列計算の一般的アルゴリズムに関しては、以下の教科書を参照されたい。

【００７５】

G. H. Golub and C. F. Van Loan "Matrix Computations" The Johns Hopkins University Press, Third edition 1996

（付記１）共有メモリ型スカラ並列計算機用逆行列の並列処理方法であって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、

該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にＬＵ分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、１つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、

を備える方法。

【００７６】

（付記２）前記共有メモリ型スカラ並列計算機は、複数のプロセッサと、該プロセッサに対応して設けられる複数のキャッシュメモリと、複数の共有メモリと、これらを通信可能なように接続する相互結合網とからなることを特徴とする付記１に記載の方法。

【００７７】

（付記３）上記方法は、Ｇａｕｓｓ－Ｊｏｒｄａｎの方法をブロック毎に並列に計算する構成としたことを特徴とする付記１に記載の方法。

（付記４）前記各ブロックを分割して並列計算するさいの分割の幅は、逆行列を求める行列の大きさと並列処理のために利用できるプロセッサ数から、並列処理しない正方形ブロックの演算量の総和が演算全体の１％程度となるように設定されることを特徴とする付記１に記載の方法。

【００７８】

（付記５）共有メモリ型スカラ並列計算機用逆行列の並列処理方法を実現させるプログラムであって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、

該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右

横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にＬＵ分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、１つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、

を備える方法を共有メモリ型スカラ並列計算機に実行させるプログラム。

【００７９】

（付記６）前記共有メモリ型スカラ並列計算機は、複数のプロセッサと、該プロセッサに対応して設けられる複数のキャッシュメモリと、複数の共有メモリと、これらを通信可能なように接続する相互結合網とからなることを特徴とする付記５に記載のプログラム。

【００８０】

（付記７）上記方法は、Ｇａｕｓｓ－Ｊｏｒｄａｎの方法をブロック毎に並列に計算する構成としたことを特徴とする付記５に記載のプログラム。

（付記８）前記各ブロックを分割して並列計算するさいの分割の幅は、逆行列を求める行列の大きさと並列処理のために利用できるプロセッサ数から、並列処理しない正方形ブロックの演算量の総和が演算全体の１％程度となるように設定されることを特徴とする付記５に記載のプログラム。

【００８１】

【発明の効果】

高性能かつスケーラビリティのある逆行列の解法を実現できる。

【図面の簡単な説明】

【図１】

本発明の実施形態が前提とする共有メモリ型スカラ計算機のハードウェア構成

を示した図である。

【図２】

本発明の実施形態の計算順序を説明する図（その１）である。

【図３】

本発明の実施形態の計算順序を説明する図（その２）である。

【図４】

本発明の実施形態の計算順序を説明する図（その３）である。

【図５】

本発明の実施形態の計算順序を説明する図（その４）である。

【図６】

本発明の実施形態の計算順序を説明する図（その５）である。

【図７】

本発明の実施形態の計算順序を説明する図（その６）である。

【図８】

本発明の実施形態の計算順序を説明する図（その７）である。

【図９】

本発明の実施形態の疑似コード（その１）である。

【図１０】

本発明の実施形態の疑似コード（その２）である。

【図１１】

本発明の実施形態の疑似コード（その３）である。

【図１２】

本発明の実施形態の疑似コード（その４）である。

【図１３】

本発明の実施形態の疑似コード（その５）である。

【図１４】

本発明の実施形態の疑似コード（その６）である。

【図１５】

本発明の実施形態の疑似コード（その７）である。

【図１６】

疑似コードの処理をフローチャートで表した図（その１）である。

【図１７】

疑似コードの処理をフローチャートで表した図（その２）である。

【図１８】

疑似コードの処理をフローチャートで表した図（その３）である。

【図１９】

疑似コードの処理をフローチャートで表した図（その４）である。

【図２０】

疑似コードの処理をフローチャートで表した図（その５）である。

【図２１】

疑似コードの処理をフローチャートで表した図（その６）である。

【図２２】

疑似コードの処理をフローチャートで表した図（その７）である。

【図２３】

疑似コードの処理をフローチャートで表した図（その８）である。

【図２４】

疑似コードの処理をフローチャートで表した図（その９）である。

【図２５】

疑似コードの処理をフローチャートで表した図（その１０）である。

【図２６】

疑似コードの処理をフローチャートで表した図（その１１）である。

【図２７】

疑似コードの処理をフローチャートで表した図（その１２）である。

【図２８】

疑似コードの処理をフローチャートで表した図（その１３）である。

【図２９】

疑似コードの処理をフローチャートで表した図（その１４）である。

【符号の説明】

１０－１～１０－ｎ　　プロセッサ

１１－１～１１－ｎ　　メモリモジュール

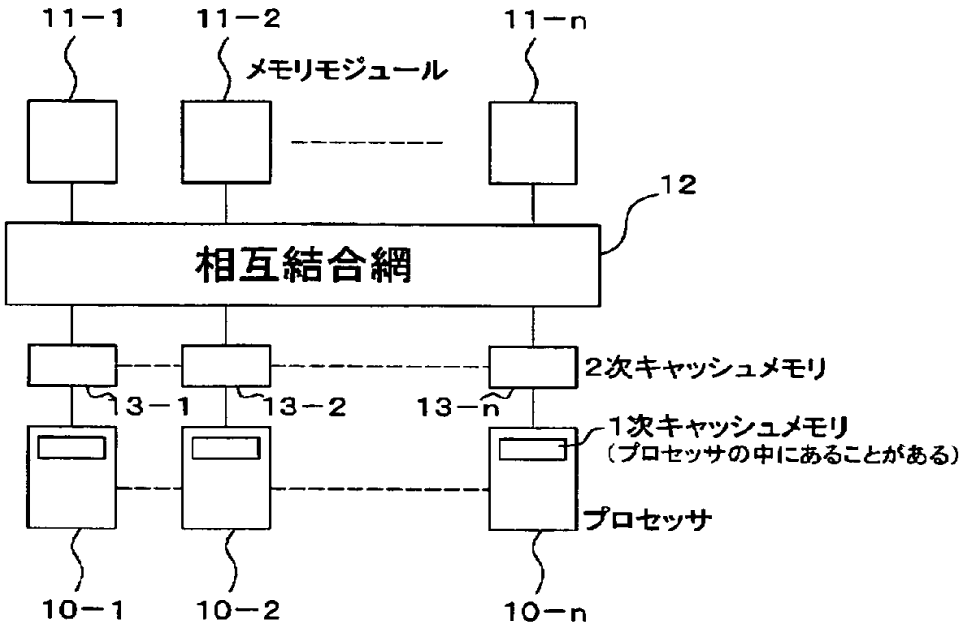
１２　　相互結合網

１３－１～１３－ｎ　　２次キャッシュメモリ

【書類名】　図面

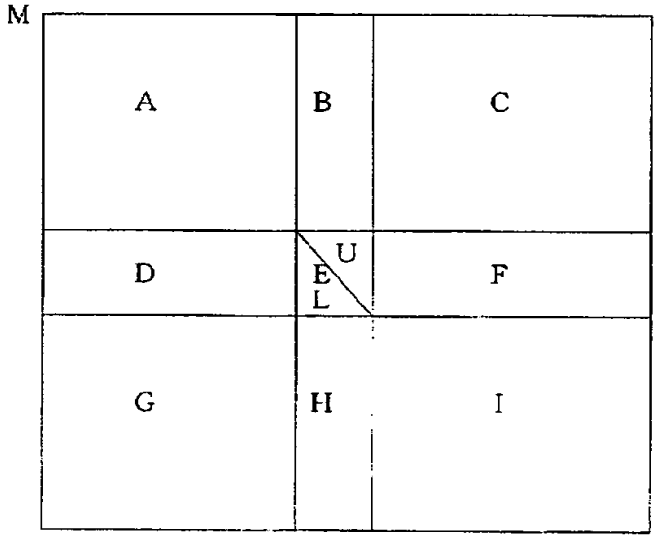
【図１】

本発明の実施形態が前提とする共有メモリ型スカラ計算機のハードウェア構成を示した図



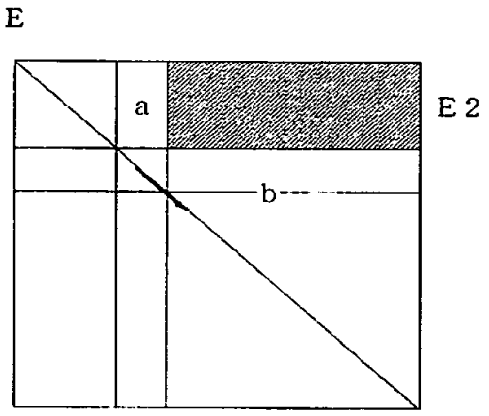
【図２】

本発明の実施形態の計算順序を説明する図（その１）



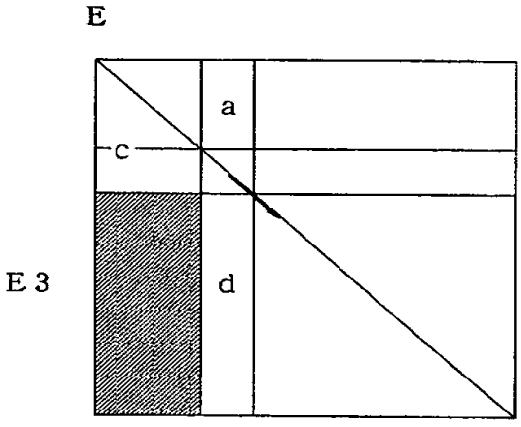
【図３】

本発明の実施形態の計算順序を説明する図（その２）



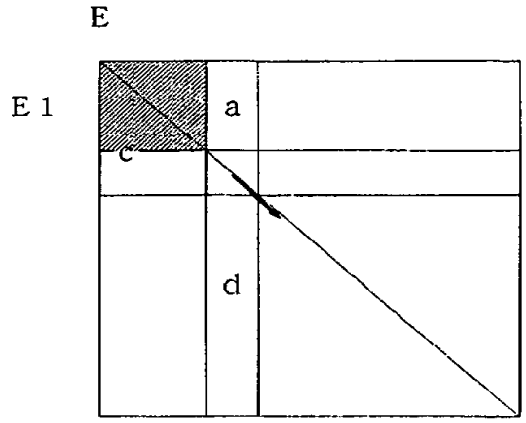
【図４】

本発明の実施形態の計算順序を説明する図（その３）



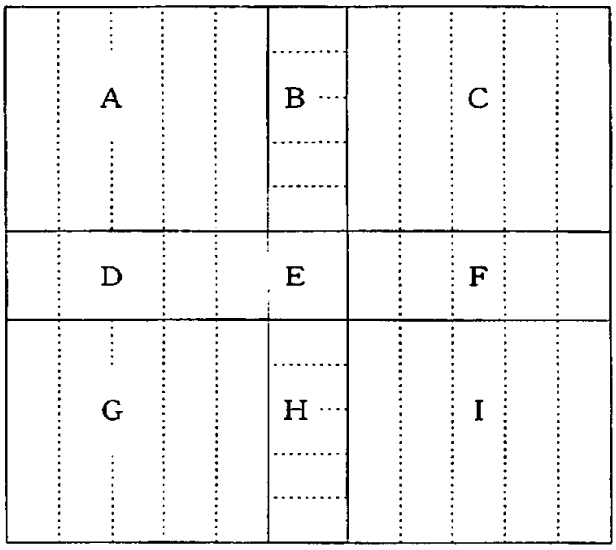
【図５】

本発明の実施形態の計算順序を説明する図（その４）



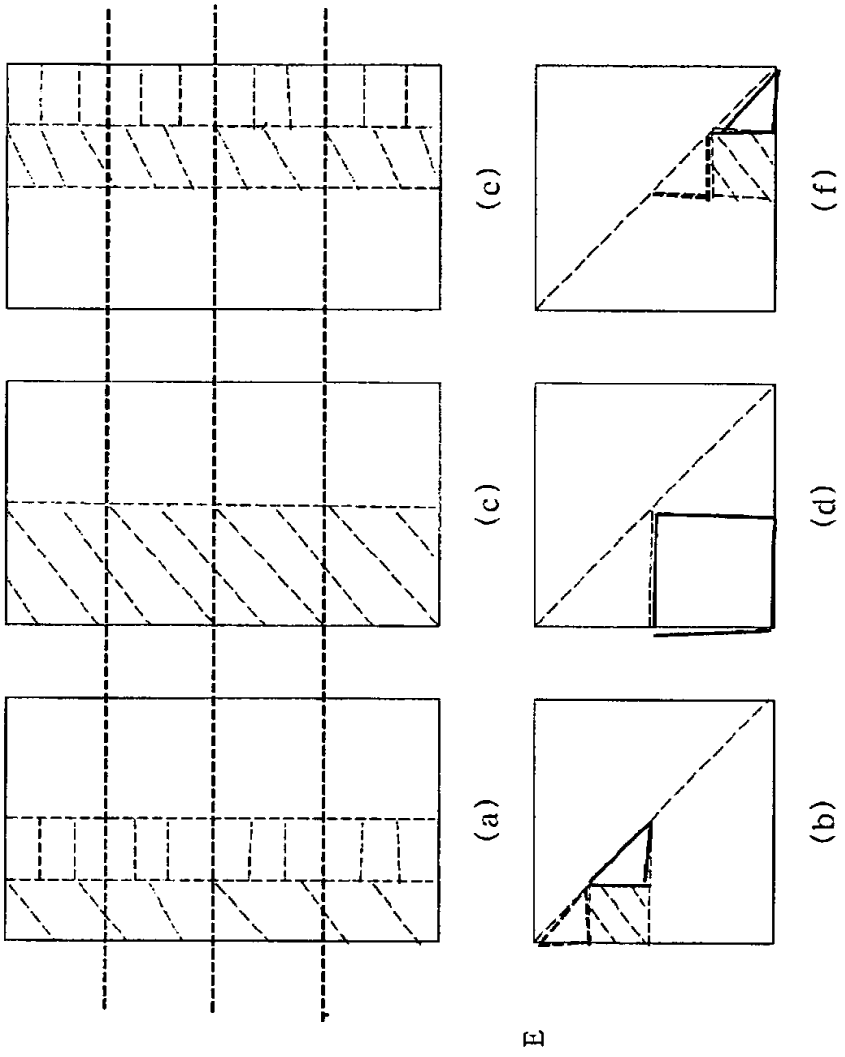
【図６】

本発明の実施形態の計算順序を説明する図（その５）



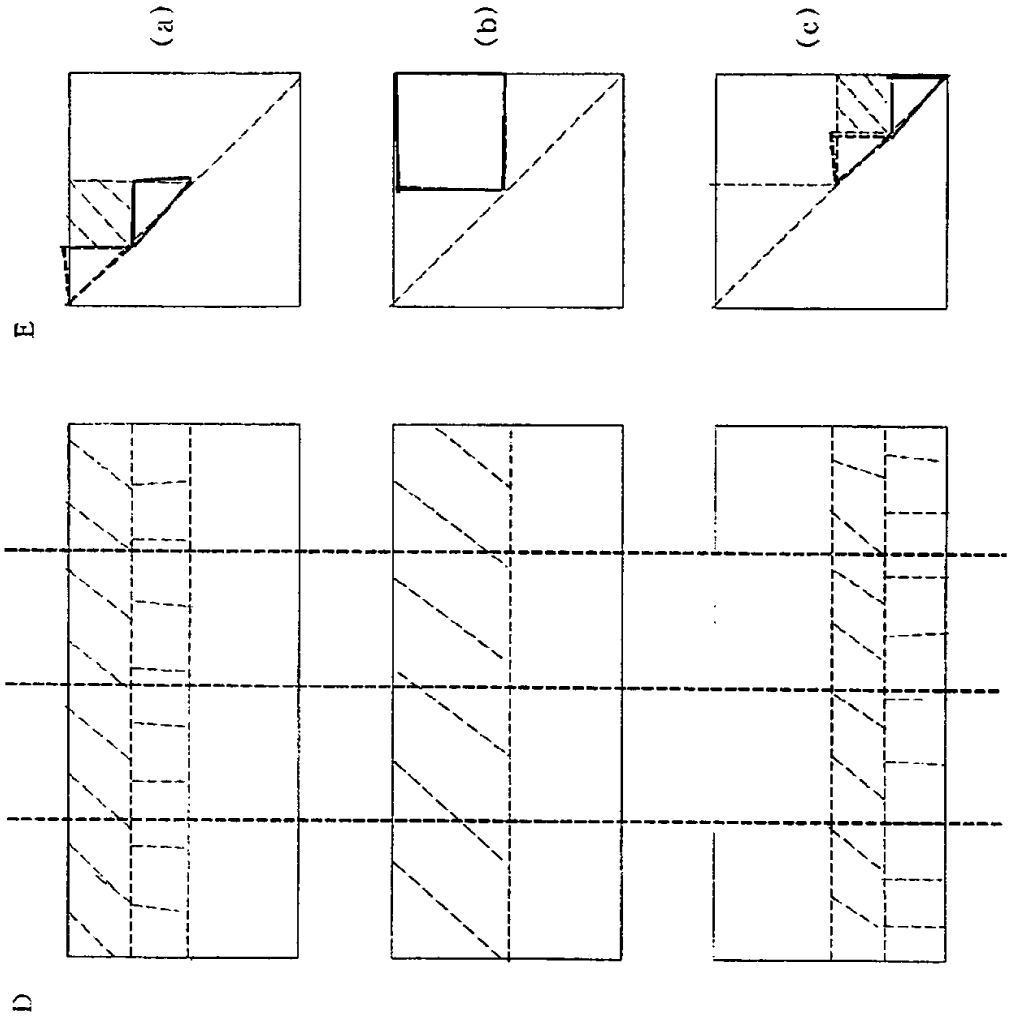
【図７】

本発明の実施形態の計算順序を説明する図（その６）



【図８】

本発明の実施形態の計算順序を説明する図（その７）



【図９】

本発明の実施形態の疑似コード（その１）

```
c       逆行列の並列アルゴリズム
        subroutine inversion(a,k,n)
        shared array a(k,n),ip(n)
        create threads
        set nothrd and numthrd
c       nothrdはスレッドの番号１～#TH、numthrd=#TH（スレッドの総数）
        iblk=ブロック幅
        nb=(n+blk-1)/blk
        do i=1,nb-1
        nbase=(nb-1)*blk
        call LU(a,k,n,nbase,blk,ip,nothrd,numthrd)
                    ! 指定されたブロックのLU分解を行い行ブロックを更新し、右下の正方
                    ! 行列を更新する演算を並列に行う。ip(nbase+1:nbase-blk)に
                    ! 行の入れ換えの情報が返却される。この部分の並列アルゴリズムは
                    ! 特願平１２－３５８２３２号に記述されているのでそれを利用。
        do i=nbase+1,nbase+blk
        if(ip(i)>i)then
        exchange(a(i,1:nbase),a(ip(i),1:nbase))
        endif
        enddo
        call update(a,k,n,nbase,blk,nothrd,numthrd)
        enddo
        nbase=(nb-1)*blk
        blklast=n-nbase
        call LU(a,k,n,nbase,blklast,nothrd,numthrd)
        call update(a,k,n,nbase,blklast,nothrd,numthrd)
        return
        end
```

【図１０】

本発明の実施形態の疑似コード（その２）

```
c     ブロック LU 分解の情報を利用して残りの部分の更新を行うルーチン
      subrourine update(a,k,n,nbase,blk,nothrd,numthrd)
      call b-update(a,k,n,nbase,blk,nothrd,numthrd)
      call d-update(a,k,n,nbase,blk,nothrd,numthrd)
      call c-update(a,k,n,nbase,blk,nothrd,numthrd)
      call a-update(a,k,n,nbase,blk,nothrd,numthrd)
      call g-update(a,k,n,nbase,blk,nothrd,numthrd)
      BARRIER SYNC
      if(nothread=1)then
c     e の更新 1
      call e-update1(a(nbase+1,nbase+1),k,n,blk)
      endif
      BARRIER SYNC
      len←(nbase+numthrd-1)/numthrd
      is←(nothrd-1)*len+1
      ie←nothrd*len
      call df-update(a(nbase+1,1),k,n,a(nbase-1,nbase-1),is,ie,blk)
      nbase2←nbase-blk
      len←(n-nbase2-numthrd-1)/numthrd
      is2←nbase2-(nothrd-1)*len+1
      ie2←nbase2-nothrd*len
      call df-update(a(nbase+1,1),k,n,a(nbase+1,nbase-1),is,ie,blk)
      BARRIER SYNC
      if(nothread=1)then
c     e の更新 2
      call e-update2(a(nbase+1,nbase+1),k,n,blk)
      endif
      BARRIER SYNC
      len←(nbase+numthrd-1)/numthrd
      is←(nothrd-1)*len+1
      ie←nothrd*len
      call bh-update(a(1,nbase+1),k,n,a(nbase+1,nbase+1),is,ie,blk)
      nbase2←nbase+blk
      len←(n-nbase2+numthrd-1)/numthrd
      is2←nbase2+(nothrd-1)*len-1
      ie2←nbase2+nothrd*len
      call bh-update(a(1,nbase+1),k,n,a(nbase-1,nbase+1),is,ie,blk)
      BARRIER SYNC
      if(nothread=1)then
c     e の更新 3
      call e-update3(a(nbase+1,nbase+1),k,n,blk)
      endif
      BARRIER SYNC
      len=(n+numthrd-1)/numthrd
      is=(nothrd-1)*len+1
      ie=min(n,nothrd*len)
      if(ip(i)>i)then
      exchange(a(is:ie,i),a(is,ie,ip(i)))
      endif
      enddo
      BARRIER SYNC

      eliminate  threads

      return
      end
```

【図１１】

本発明の実施形態の疑似コード（その３）

```
c     ブロックＢの更新
      subroutine b-update(a,k,n,nbase,blk,nothrd,numthrd)
      shared array a(k,n)
      len←(nbase+numthrd-1)/numthrd
      is1←(nothrd-1)*len
      ie1←nothrd*len
      iof=nbase
      a(is:ie,iof+1:iof+blk)
      ←a(is:ie,iof-1:iof+blk)*TRU-U(iof+1:iof+blk,iof-1:iof+blk)^-1
        ! TRU-U  対角要素を１.０にした上三角行列
      return
      end

c     ブロックＤの更新
      subroutine d-update(a,k,n,nbase,blk,nothrd,numthrd)
      shared array a(k,n)
      len←(nbase+numthrd-1)/numthrd
      is1←(nothrd-1)*len
      ie1←nothrd*lenis1
      iof=nbase
      a(is:ie,iof+1:iof+blk)
      ←TRL(iof+1:iof+blk,iof-1:iof+blk)^-1*a(is:ie,iof-1:iof+blk)
           ! TRL は下三角行列
      return
      end
```

【図１２】

本発明の実施形態の疑似コード（その４）

```
c       ブロックＣの更新
        subroutine c-update(a,k,n,nbase,blk,nothrd,numthrd)
        shared array a(k,n)
        nbase2←nbase+blk
        iof=nbase
        len←(n-nbase2+numthrd-1)/numthrd
        is2←nbase2+(nothrd-1)*len+1
        ie2←nbase2+nothrd*len
        a(1:iof,is2:ie2)
        ←a(1:iof,is2:ie2)
        -a(1:iof,iof+1:iof+blk)*a(iof+1:iof+blk,is2:ie2)
        return
        end

c       ブロックＡの更新
        subroutine a-update(a,k,n,nbase,blk,nothrd,numthrd)
        shared array a(k,n)
        len←(nbase+numthrd-1)/numthrd
        is2←(nothrd-1)*len
        ie2←nothrd*len
        iof=nbase
        a(1:iof,is2:ie2)
        ←a(1:iof,is2:ie2)
        -a(1:iof,iof+1:iof+blk)*a(iof+1:iof+blk,is2:ie2)
        return
        end
```

【図１３】

本発明の実施形態の疑似コード（その5）

```
c       ブロックＧの更新
        subroutine g-update(a,k,n,nbase,blk,nothrd,numthrd)
        shared array a(k,n)
        len←(nbase+numthrd-1)/numthrd
        is2←(nothrd-1)*len
        ie2←nothrd*len
        iof←nbase+blk
        a(iof+1:n,is2:ie2)
        ←a(iof+1:n,is2:ie2)
        -a(iof+1:n,iof+1:iof+blk)*a(iof+1:iof+blk,is2:ie2)
        return
        end

c       ブロックＥの最初の更新
        subroutine e-update1(s,k,n,blk)
        shared array s(k,n)

        do i=1,blk
        s(1:i-1,i+1:blk)←s(1:i-1,i+1:blk)-s(1:i-1,i)*s(I,i+1:blk)
        enddo
        return
        end

c       ブロックＥの２回目の更新
        subroutine e-update2(s,k,n,blk)
        shared array s(k,n)

        do i=1,blk
        tmp←1.0/s(i,i)
        s(i,1:i-1)←tmp*s(i,1:i-1)
        s(i+1:blk,1:i-1)←s(i+1:blk,1:i-1)-s(i,1:i-1)*s(i+1:blk,i)
        s(i+1:blk,i)←-s(i+1:blk,i)*tmp
        s(i,i)←tmp
        enddo
        return
        end
```

【図１４】

本発明の実施形態の疑似コード（その６）

```
c       ブロックＥの最後の更新
        subroutine e-update3(s,k,n,blk)
        shared array s(k,n)

        do i=1,blk
        s(1:i-1,1:i-1)←s(1:i-1,1:i-1)-s(1:i-1,i)*s(i,1:i-1)
        s(1:i-1,i)←-s(1:i-1,i)*a(i,i)
        enddo
        return
        end

c       ブロックＤおよびＦの更新
        subroutine df-update(a,k,n,s,is,ie,len)
        shared array a(k,*),s(k,*)
        if(len<10)then
        do i=1,len
        a(1:i-1,is:ie)←a(1:i-1,is:ie)-s(1:i-1,i)*a(i,is:ie)
        enddo
        else
        if(len>=32 or len<=20)then
        len1←len/2
        len2←len-len1
        else
        len1←len/3
        len2←len-len1
        endif
        call df-update(a,k,n,s,is,ie,len1)
        a(1:len1,is:ie)
        ←a(1:len1,is:ie)-s(1:len1:len1+1:len)*a(len1+1:len,is,ie)
        call df-update(a(len1+1,1),k,n,s(len1+1,len1+1),is,ie,len2)
        endif
        return
        end
```

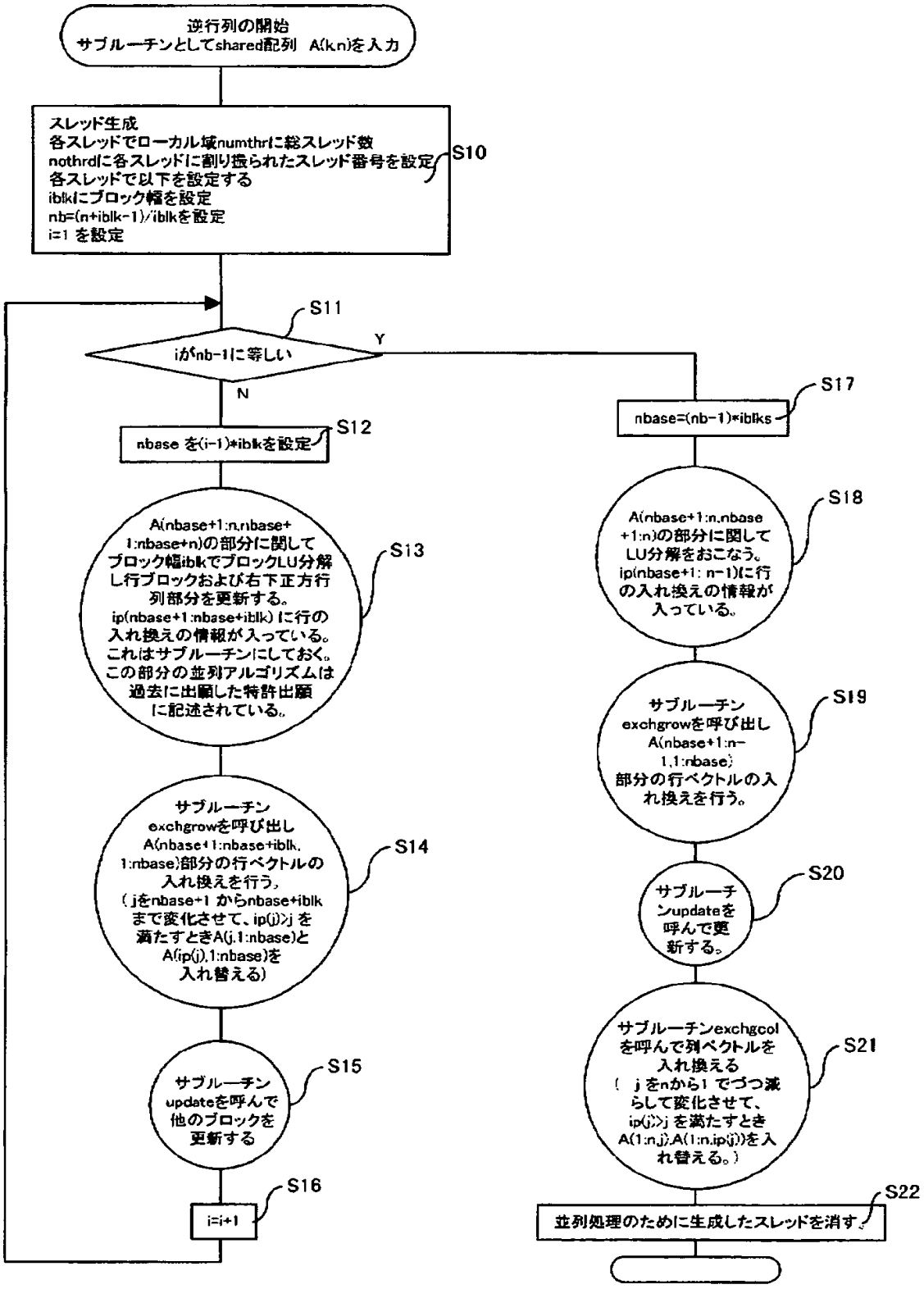
【図１５】

本発明の実施形態の疑似コード（その７）

```
c       ブロックＢおよびＨの更新
        dubroutine bh-update(a,k,n,s,is,ie,len)
        shared array a(k,*),s(k,*)
        if(len<10)then
        do i=1,len
        a(is:ie,i)←-a(is:ie,i)*s(i,i)
        a(is:ie,i)←a(is:ie,i)-a(is:ie,i-1:len)*s(i+1:len,i)
        enddo
        else
        if(len>=32 or len<=20)then
        len1←len/2
        len2←len-len1
        else
        len1←len/3
        len2←len-len1
        endif
        call bh-update(a,k,n,s,is,ie,len1)
        a(is:ie,1:len1)←a(is:ie,1:len1)
                        -a(is:ie,len1+1:len)*s(len1+1:len,1:len1)
        call bh-update(a(1,len1+1),k,n,s(len1+1,len1+1),is,ie,len2)
        endif
        return
        end
```

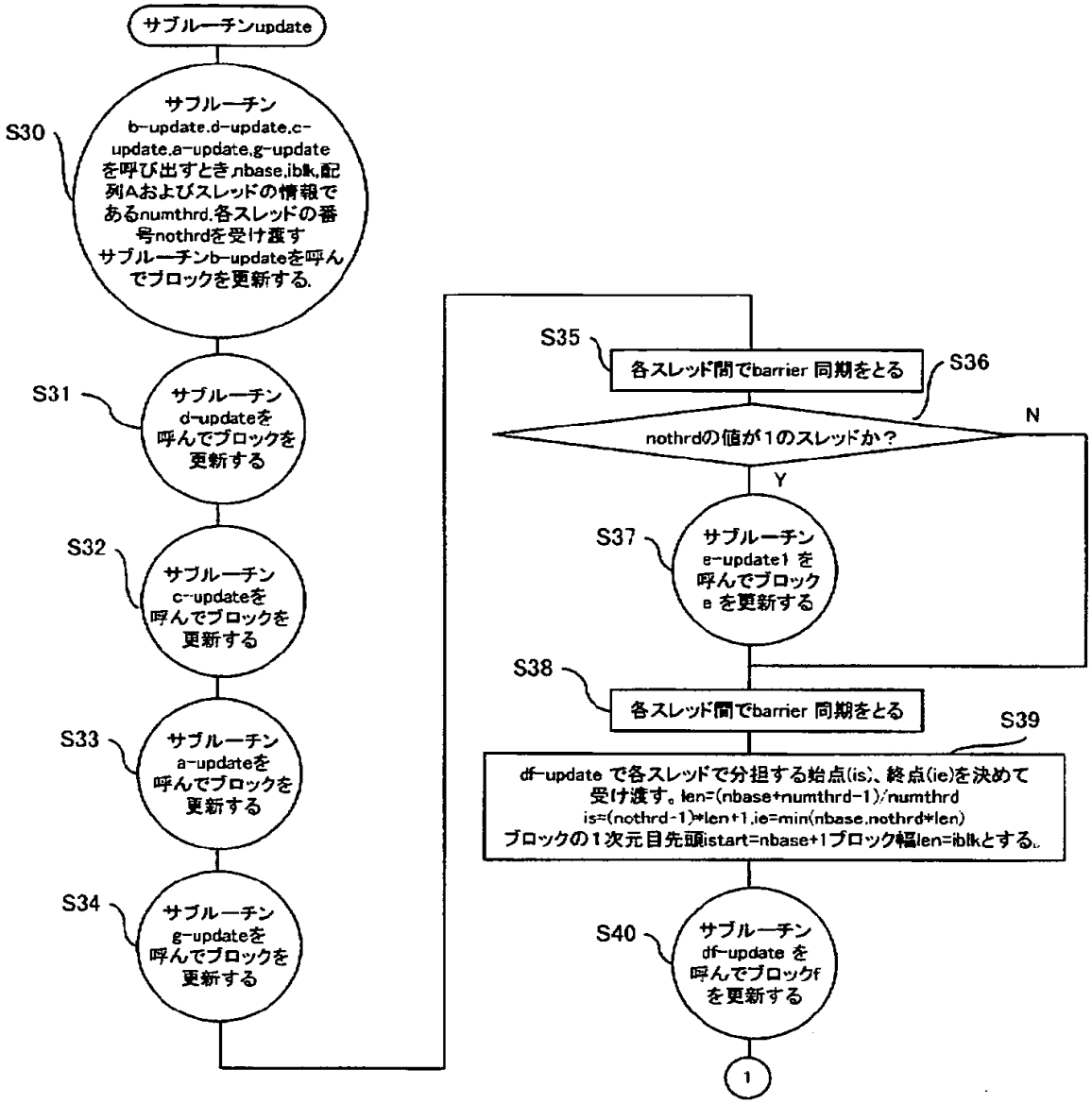
【図１６】

疑似コードの処理をフローチャートで表した図（その１）



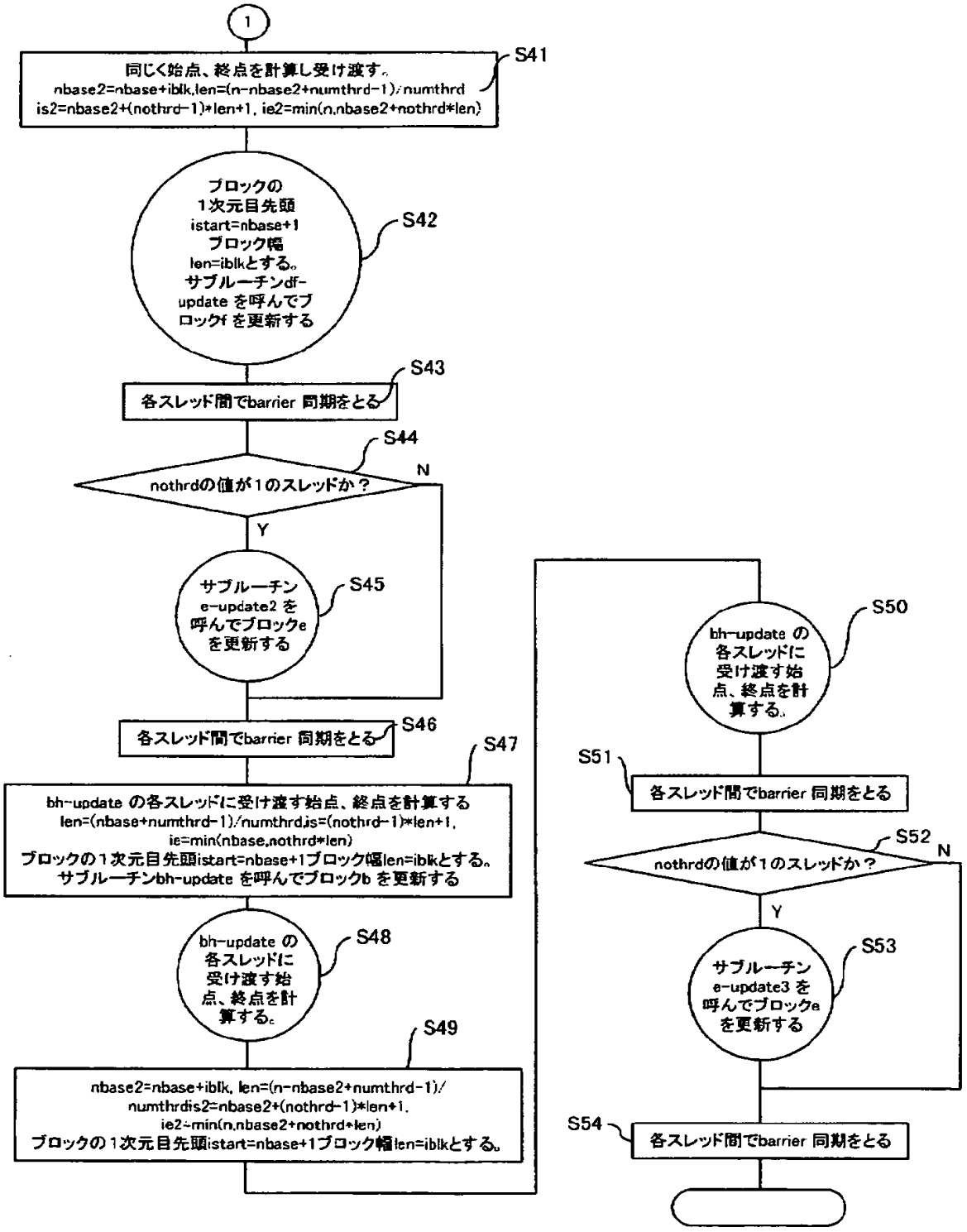
【図１７】

疑似コードの処理をフローチャートで表した図（その２）



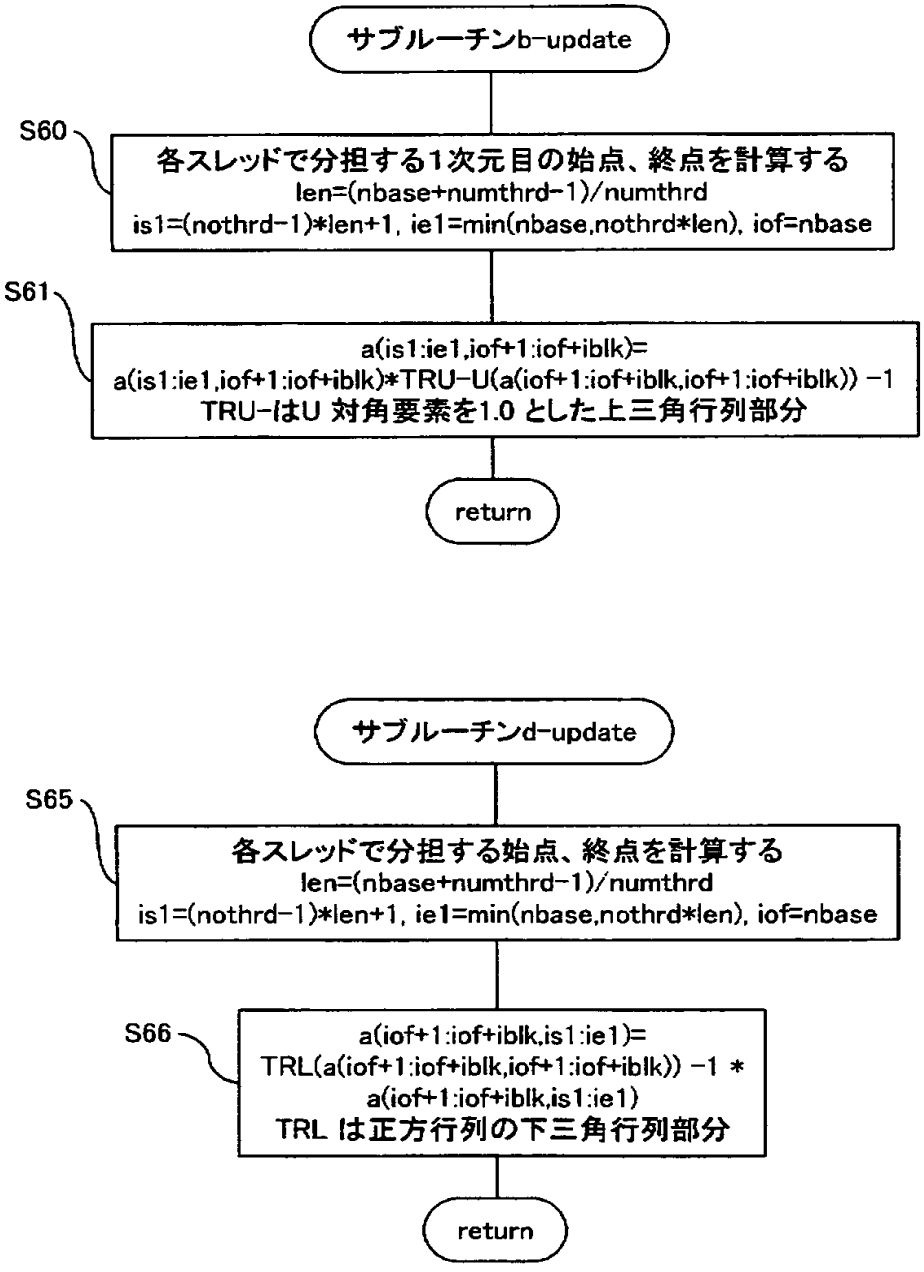
【図１８】

疑似コードの処理をフローチャートで表した図（その3）



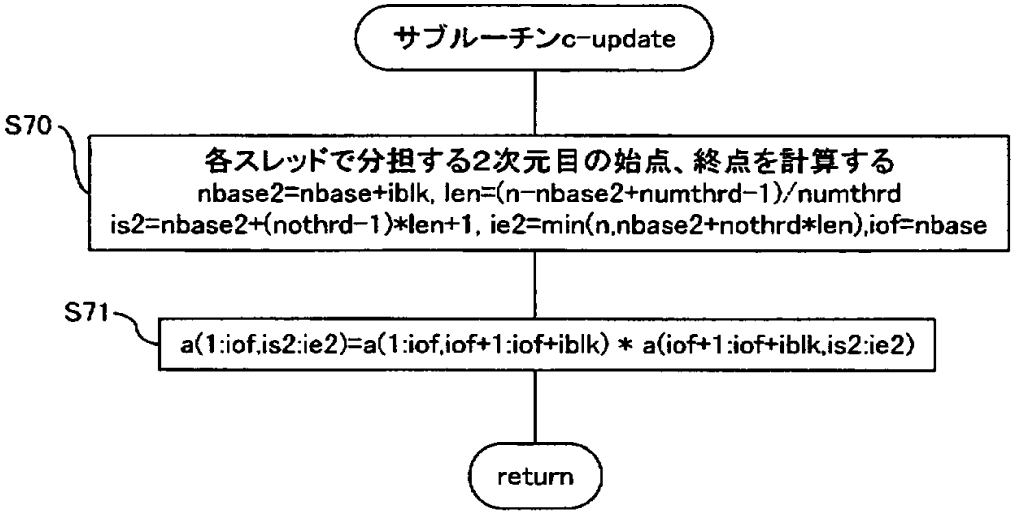
【図１９】

疑似コードの処理をフローチャートで表した図（その４）



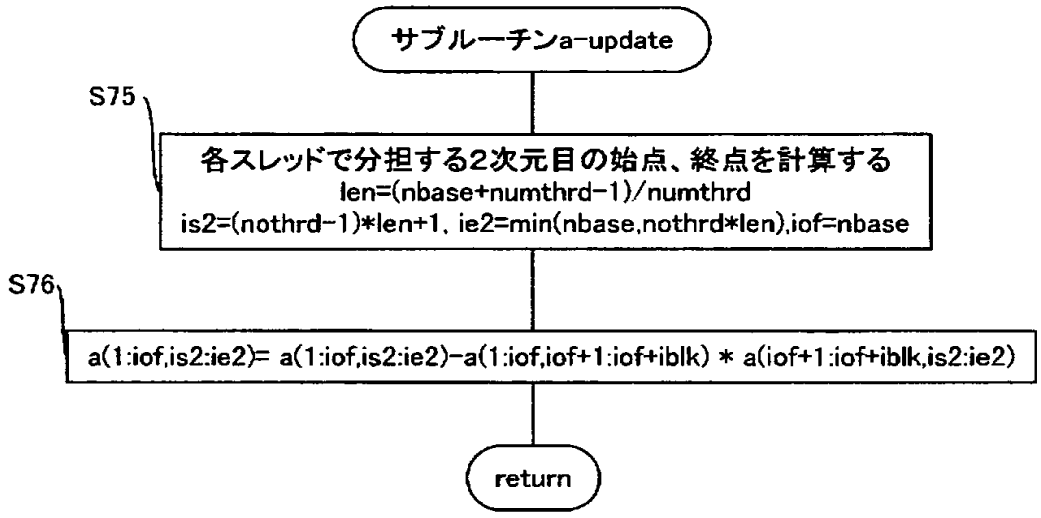
【図２０】

疑似コードの処理をフローチャートで表した図（その5）



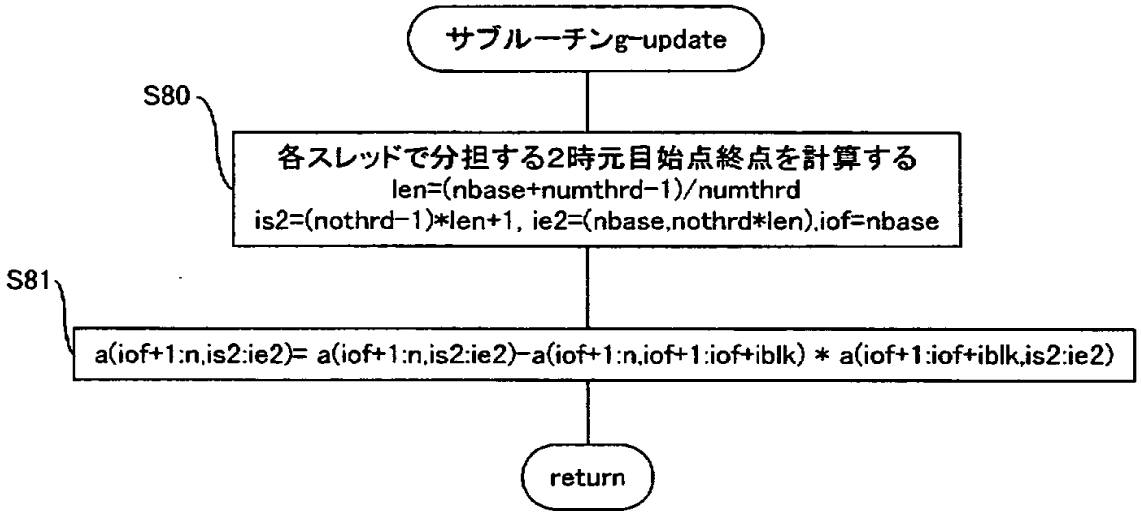
【図２１】

疑似コードの処理をフローチャートで表した図（その６）



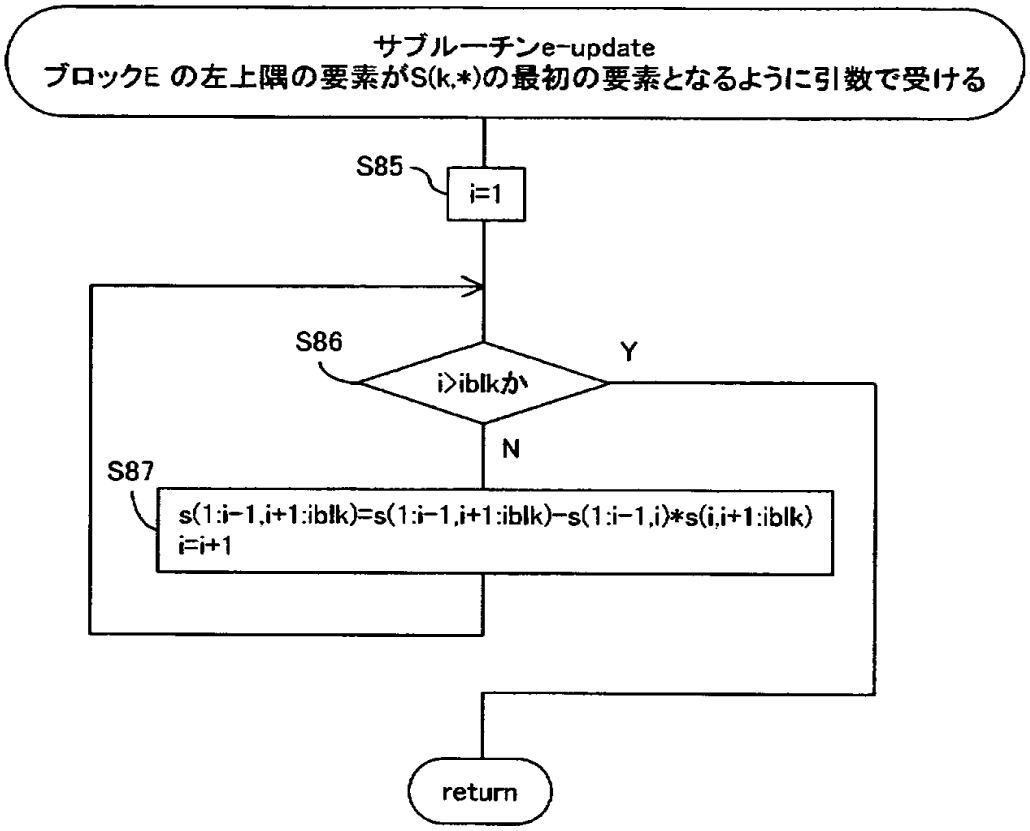
【図２２】

疑似コードの処理をフローチャートで表した図（その7）



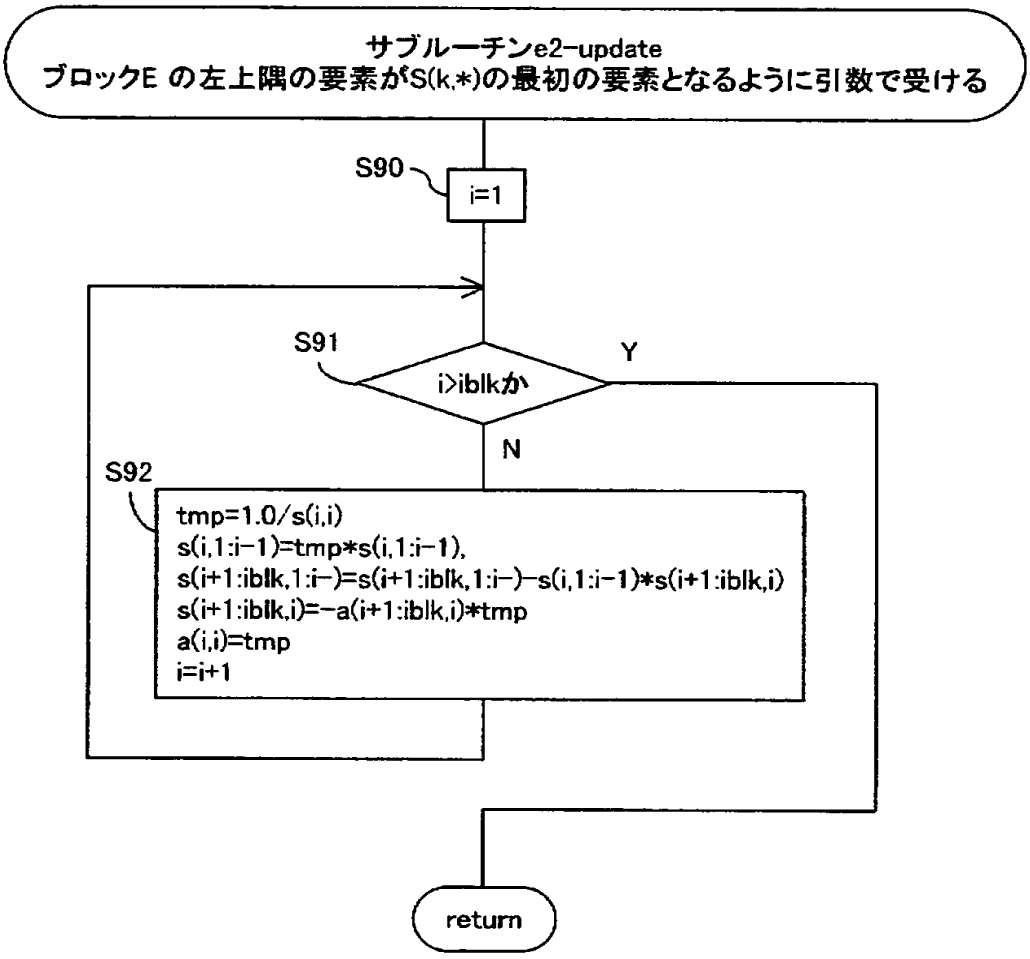
【図２３】

疑似コードの処理をフローチャートで表した図（その８）



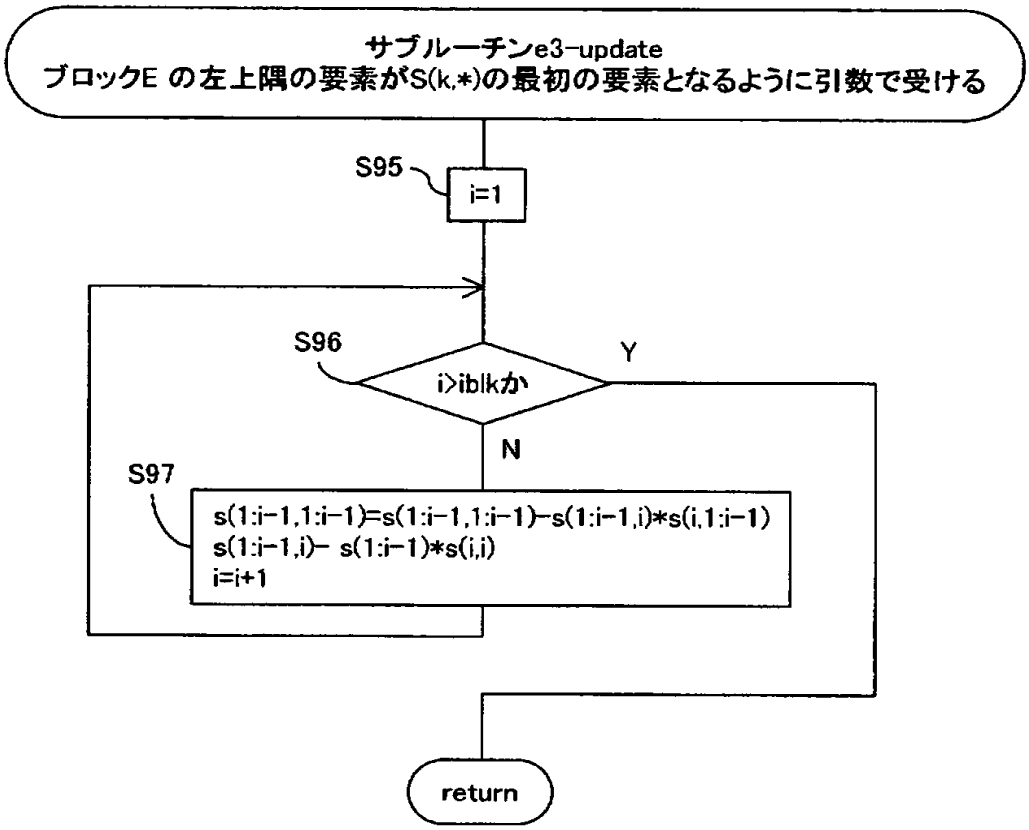
【図２４】

疑似コードの処理をフローチャートで表した図（その９）



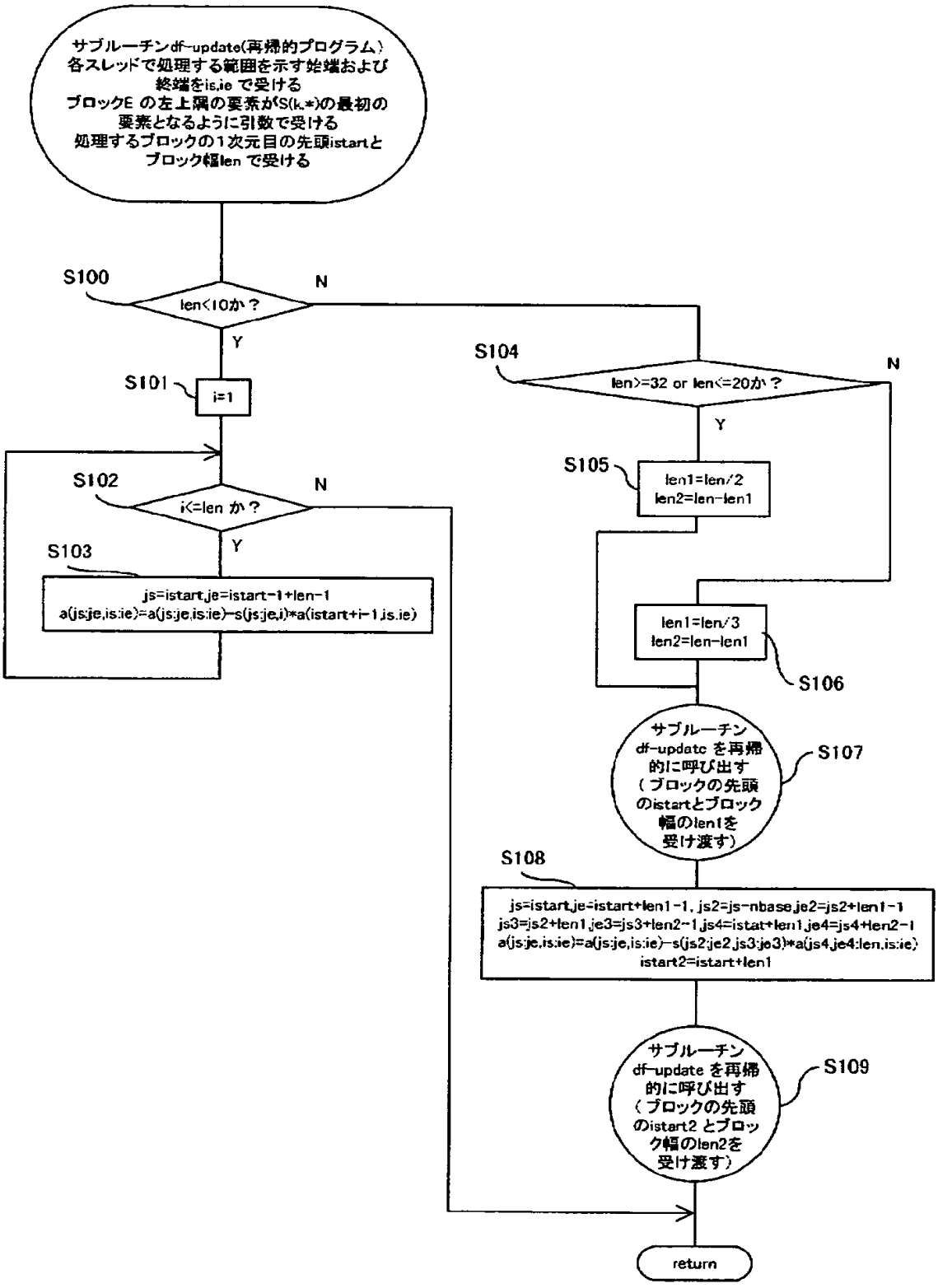
【図２５】

疑似コードの処理をフローチャートで表した図（その10）



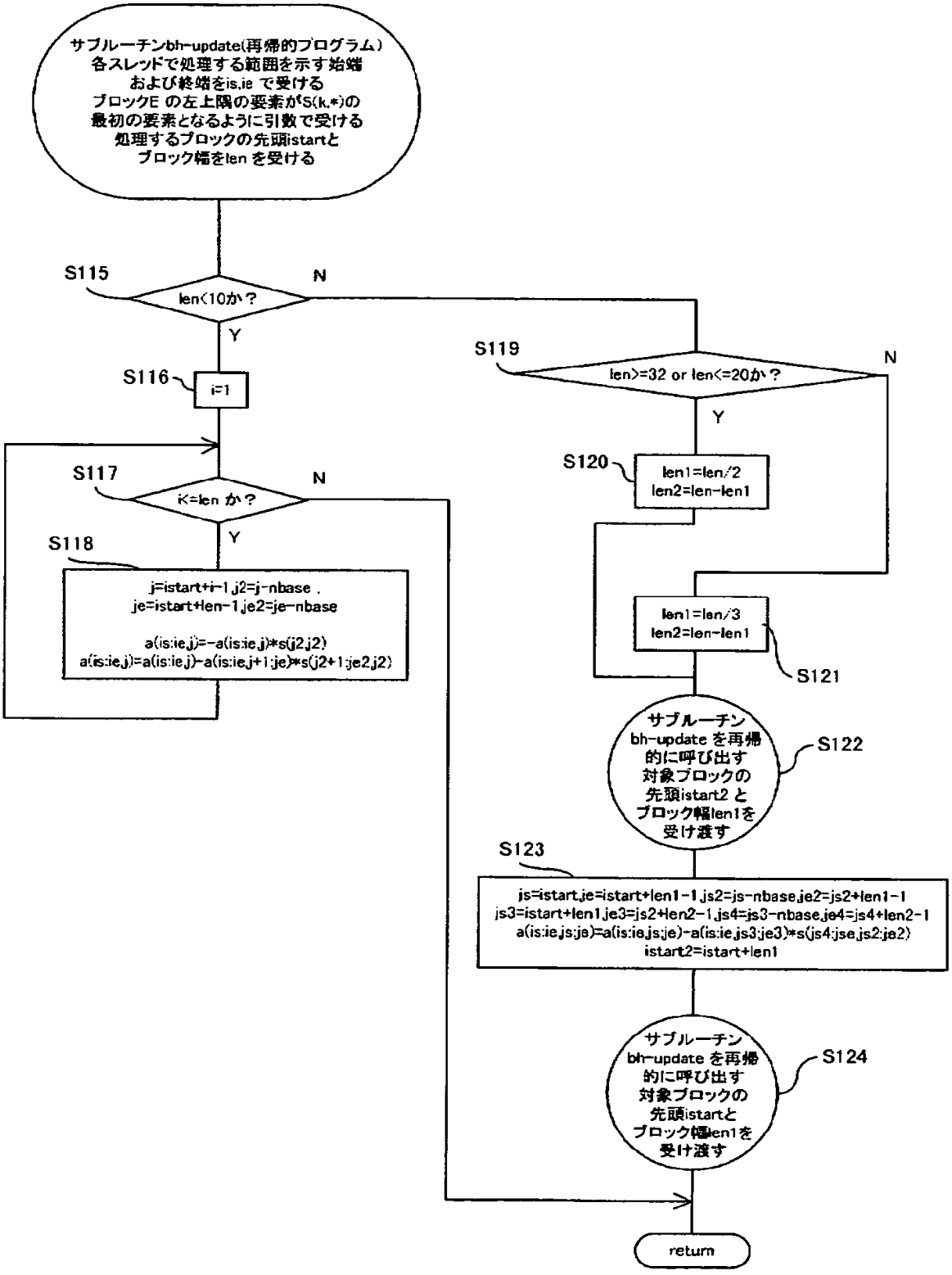
【図２６】

疑似コードの処理をフローチャートで表した図（その11）



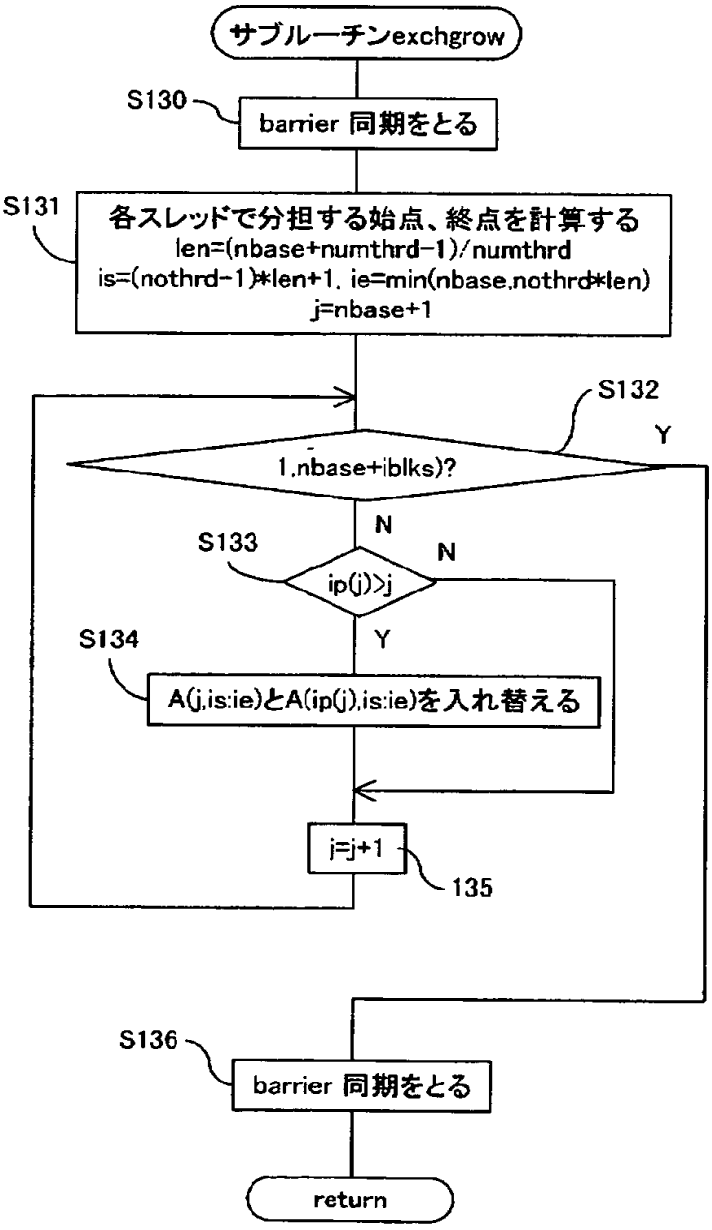
【図２７】

疑似コードの処理をフローチャートで表した図（その12）



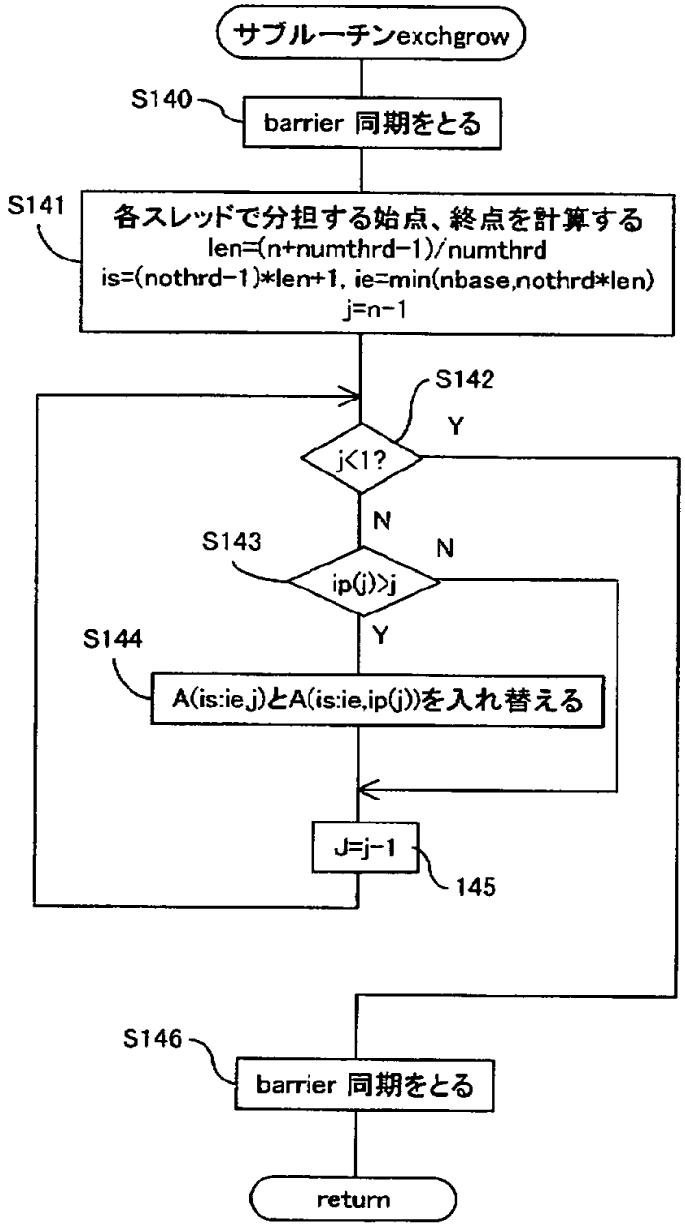
【図２８】

疑似コードの処理をフローチャートで表した図（その13）



【図２９】

疑似コードの処理をフローチャートで表した図（その１４）



【書類名】　　要約書

【要約】

【課題】共有メモリ型スカラ計算機において、高速に逆行列の演算が行える演算の並列処理方法を提供する。

【解決手段】まず、ブロックＥとＨＬＵ分解する。次に、ブロックＢをブロックＥの上三角部分を使って更新し、ブロックＥの下三角部分を使って、ブロックＤを更新する。このとき、ＬＵ分解によって、ブロックＦとＩが更新済みである。そして、ブロックＢ、Ｄ、Ｆ、Ｈを使って、ブロックＡ、Ｃ、Ｇ、Ｉを更新し、ブロックＥの上三角部分を更新し、最後にブロックＤ、Ｆを更新する。そして、ブロックＥを２回目の更新をして、その結果を用いて、ブロックＢ、Ｈを更新する。最後に、ブロックＥの最終的な更新をして、ピボットの入れ替え処理を終えて処理を終了する。これらのブロックの処理は、複数のスレッドに分割され、平行して行われる。

【選択図】　　図２

特願２００３－０７４５４８

出願人履歴情報

識別番号　［０００００５２２３］

１．変更年月日　１９９０年　８月２４日
［変更理由］　新規登録
住　所　神奈川県川崎市中原区上小田中１０１５番地
氏　名　富士通株式会社

２．変更年月日　１９９６年　３月２６日
［変更理由］　住所変更
住　所　神奈川県川崎市中原区上小田中４丁目１番１号
氏　名　富士通株式会社